Reengineering and extending the Agents in Grid Ontology

Paweł Szmeja[1], Katarzyna Wasielewska[1], Maria Ganzha[1], Michał Drozdowicz[1], Marcin Paprzycki[1], Stefka Fridanova[2], and Ivan Lirkov[2]

[1] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
{pawel.szmeja, katarzyna.wasielewska, drozdowicz}@gmail.com,
{maria.ganzha, marcin.paprzycki}@ibspan.waw.pl
[2] Institute of Information and Communication Technologies Bulgarian Academy of Sciences, Sofia. Bulgaria {stefka, ivan}@parallel.bas.bg

Abstract. Ontology engineering, despite considerable progress, is still relatively new and dynamically evolving discipline. As a result, the *universal* standards for creating and/or editing an ontology, have not been established. This leads to problems with reusing and updating existing ontologies. It also makes writing an ontology from scratch seem like a good idea. The aim of this paper is two-fold. First, to discuss key issues encountered during re-engineering of an existing ontology. Second, to show how the good practices of ontology development were applied to model the area of computational linear algebra. Here, special attention is paid to the application of this ontology in the user support system.

1 Introduction

The context for this paper is provided by the *Agents in Grid* project (*AiG*; [3, 6, 9]), which aims at development of an agent-based infrastructure for intelligent resource management in the Grid. The *AiG* project combines software agents and semantic data processing. Specifically, all knowledge in the system is stored in / represented as an ontology, while communication protocols utilize messages with ontological content ([5]). During the development of the system, three ontologies were designed to provide concepts necessary for: (i) resource and Grid structure description, (ii) contract and requirements specification, and (iii) content of messages exchanged in the system.

As the development of the system progressed, the ontological structures started to become complex (ontology consisting of 401 entities). Furthermore, when reasoning moved beyond simplistic examples (ontologies with a few concepts), we have been confronted with recurring errors generated by the reasoners. Therefore, the ontology reengineering became a necessity.

2 *AiG* ontology reengineering

The original *AiG* grid ontology was created on the basis of the *Core Grid Ontology* (*CGO*; 217 entities). The *CGO* was extended (adding 88 entities) and

modified to match the needs of the *AiG* project (see, [5, 9]). During this process, features identified as most problematic, from the point of view of the *AiG* project, have been modified. Furthermore, constraints and messaging ontologies have been created (96 additional entities). However, no major "checking" of the *CGO* has been performed at that stage. Let us, therefore, summarize key issues encountered when such check was performed for the complete set of *AiG* ontologies.

2.1 Documentation standards

It is crucial that the ontology is intuitive enough and that the intended use of its entities is clear. In OWL ([1]), this can be achieved through proper documentation, clear naming scheme, and overall consistency. The ontology should be uniform when representing the real world concepts and objects as OWL classes, properties and individuals. However, as we have found, the original *CGO* had problems in this area, and some of them carried over to the *AiG* 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 the 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.

Here, an example is the *operatingSystem* property that not only conflicted the naming scheme (it applied to properties such as *hasCPU* and *hasFileSystem*), but also could be easily confused with the *OperatingSystem* class. Recall that in OWL, IRIs should be unique in the scope of an ontology, *regardless* of the type of the entity. To solve the problem, the property has been renamed to *hasOperatingSystem*. The remaining (similar) problems have also been fixed.

Proper documentation should help reusability, e.g. by explaining how the ontology is intended to be used. While the OWL annotations can be used as the documentation, this was not the case with the *CGO* (only 7 classes had commenting annotations). The *AiG* ontologies are constantly updated with annotations that are to serve both as guidelines for the users and as reminders for the developers. In the future, we plan to use annotations in the dynamic user interface (see, [4] for more details). Here, the GUI, in addition to adjusting to the ontology structure, would also display information (contained in annotations) to explain to the users (a) the entities in the ontology, and (b) their intended meaning.

2.2 Ontology hierarchy

Recall, that the *AiG* grid ontology *extended* the *CGO* ontology to better fit the needs of the *AiG* system. When analyzing the interplay between these two ontologies it becomes clear that they are "conceptually" on the same level. Entities defined in the *CGO* could be transferred to the *AiG* grid ontology and vice-versa. This could be done without disturbing the main ideas underlying

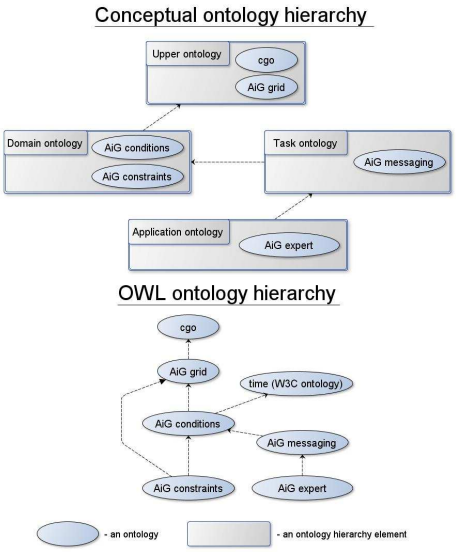

Fig. 1. Hierarchy of ontologies in *AiG*

both ontologies and the *AiG* system. Furthermore, this could be done without impacting the *work* of the *AiG* system. However, this means that the *AiG* grid ontology must be used together with the *CGO*. This demonstrates a more general issue that is rarely discussed. The typical ontology hierarchy does not take into account the fact that, on each conceptual level, there may exist multiple *ontology files*. In the *AiG* system, the ontological base consists of the *CGO* and the *AiG* ontologies, with the messaging (*AigMessagingOntology*), the contract constraints (*AiGConstraintsOntology*), and the domain (expert, *AiGExpertOntology*) ontologies placed deeper in the hierarchy. Figure 1 presents the relation of ontology files within the actual ontology hierarchy.

The reengineering that started with the *CGO* involved changes that had to be immediately reflected in the *AiG* grid ontology, in order to preserve the connection between them, and to prevent introducing (new) errors. An example of how the original *CGO* was unsuitable for being extended 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 states that every CPU needs a defined property *clockSpeed*. The latter restricts the *clockSpeed* to the CPUs *only*. The *AiG* grid ontology introduced the GPUs that had also 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 technically correct, such inference was against our intentions. 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 "too specific" upper ontology. When narrowing the domain, one might come to a *false* conclusion that the CPUs are the only objects characterized by clock speed. Furthermore, it serves no purpose in the scope of the ontology itself. Associating clock speed with CPUs is the *suggested* use, not the *only use* and, therefore, it should be put in annotations. Similarly, the extension of the domain (in the original *AiG* grid ontology) was incorrect. Note that, if we added some *Accelerated Processing Unit* to the ontology, we would face the same problem, and could end with three properties, each representing clock speed, but for different entities and with differently scoped domains.

During the reengineering, we put the *hasClockSpeed* property in place of the *clockSpeed*, with the domain specification set to *Thing* or *CPU* or *GPU*. In other words, we use the *CGO* defined property (not defining our own) and add suggested use in both annotation and domain definition (without narrowing it down). In this way we fixed similar problems associated with other properties.

2.3 Cleaning conceptual inconsistencies

A number of entities with the same intended meaning were present (at the same time) in both in the *CGO* and the *AiG* ontologies. For example, both ontologies included a *CPU* class. They had a different IRI base and a different definition (e.g. one had the *clockSpeed* property in the definition). Individuals that should belong to a *single CPU* class were divided between them. As a result, the reasoning about individuals in the *CPU* class never gave a complete result (unless done in the scope of the IRI bases of both ontologies and then combined). As a consequence, multiple reasoners (tried in the system) had problems with creating an inferred hierarchy, or classifying the ontology. 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.

We have found that this problem resulted from a misconception (or a bug) in earlier versions of the Protegé platform that assumed that classes of newly created individuals belong to the active ontology, and asserted their existence (if they were not present). The newer versions of Protegé do not suffer from this problem. This shows that growth of knowledge about "ontologies in practice" leads to development of better tools, but leaves behind ontologies with limited usability. All such problems were fixed, which also solved the reasoning errors.

3 Lessons learned

Here, let us note that literature considers mostly ontology creation, rather than long-term (re)use (see, for instance [2]). Furthermore, ongoing research concerns ontology merging, alignment, mapping, but almost nothing concerns "software

engineering like" principles for ontology re-use. This being the case let us summarize the most important lessons learned from our work.

First, 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 "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.

Second, ontologies are meant to be reused. Thus, it is crucial to clearly communicate their intended use (e.g. by providing complete annotations and adhering to the naming standards). As seen in the examples above, this can help prevent misusing a concept or (re)defining it more times than intended.

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

4 Adding a domain ontology

During the development of the *AiG* system, the need to add a new (created from scratch) ontology arose. Note that the *AiG* system is to provide support beyond the functionalities found in the existing Grid middlewares. Specifically, ontological representation of domain knowledge is to be a part of the decision support provided to the user. For instance, it should help the user to choose optimal algorithm and/or resource to solve her problem. Hence, this is another attempt (using modern tools) to achieve goals summarized in [7, 8]. While work completed in 1990's did not gain traction, we believe that with help of ontologies and semantic data processing we may have more success. As a starting point, we have focused on computational linear algebra. The ontology under development is extending the existing *AiG* ontologies, and created taking into account the lessons learned from the reengineering of the *AiG* ontologies. The main goal of the *AiGExpertOntology* is to provide concepts necessary to capture three aspects of the domain: (i) problems to be solved, (ii) algorithms to solve them, (iii) objects that these algorithms operate on. Additionally, classes *DomainExpert* and *ExpertOpinion* where introduced to represent experts knowledge (recommendations) allowing matching of problems and algorithms. Therefore, the *ExpertOpinion* class has property *hasRecommendedResource*, which points to a resource that is most suitable for solving a specific problem (according to the expert). Obviously, resources originate from the *AiG* ontology. Let us now present the preliminary hierarchy of problems in computational linear algebra (Figure 2). Here, we distinguished five types of problems represented with OWL classes: eigenproblem that can be further categorized into eigenvalue or eigenvector problem, least squares problem, solution of a system of linear equations, and calculation of a matrix norm.

The second part is the *Algorithm*; a superclass for classes (in Figure 3, we present a fragment of this hierarchy) representing algorithms that can be used to solve problems from Figure 2, for a given input data (represented in the

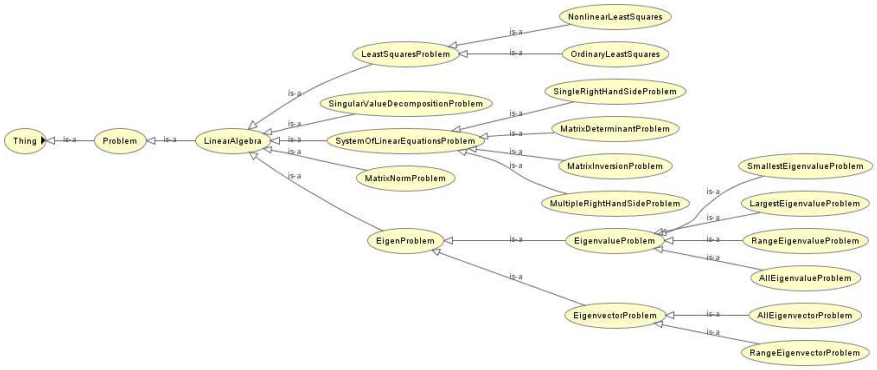

Fig. 2. Hierarchy of problems in *AiGExpertOntology*

Matrix class). This part of the ontology is going to be most complex and is being developed based on domain expert knowledge.

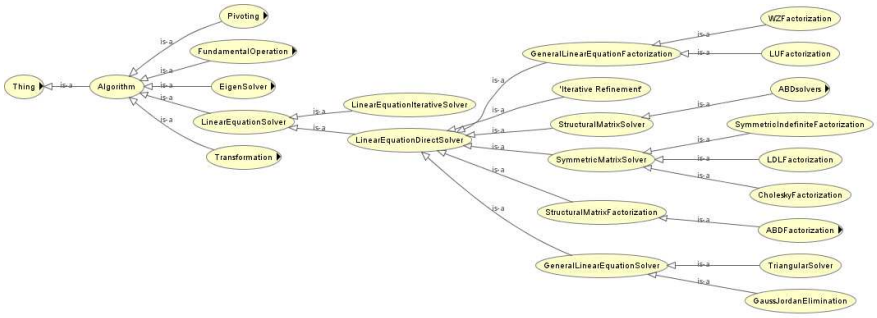

Fig. 3. Part of hierarchy of algorithms in *AiGExpertOntology*

Finally, we develop *Matrix* and *MatrixProperty* classes (Figure 4) and the property *hasMatrixProperty* that defines their relationship. The *MatrixProperty* class is a superclass for a hierarchy of properties that describe the matrix (e.g. symmetricity, density, structure, etc.). Obviously, in Figure 4 we present only fragments of the ontology that is being extended on the basis of expert opinions.

To illustrate how we plan to use the *AiGExpertOntology* ontology, let us consider a scenario, where the user is looking for a team to commission a job. Here, she could specify only requirements for resources needed to execute the job (assuming that she is certain about her needs). However, she could also indicate an individual of a subclass of the *Problem* class (Figure 2), e.g. *SystemOfLinearEquationsProblem*. Such individual may have (optional) properties that specify

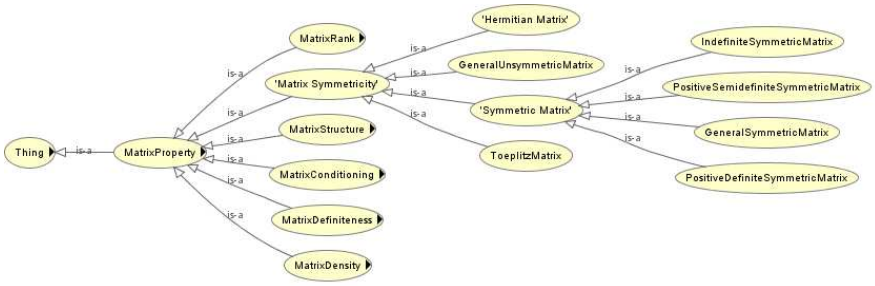

Fig. 4. Part of hierarchy of matrix properties in *AiGExpertOntology*

(i) the input, e.g. individual of class *Matrix* with values of *hasProperty* being individuals of classes *PositiveDefiniteMatrix* and *SymmetricMatrix*, and (ii) algorithm, e.g. individual of class *CholeskyFactorization*. The first use case is to validate user request for a resource against resources recommended by experts for a combination of problem, input data, and algorithm. User's resource specification is evaluated against experts suggestions using Saaty's Analytical Hierarchy Process (AHP) for multicriterial assessment. The way to combine ontologies and the AHP method was introduced in [10].

Here, two use cases can be distinguished. First, when user requirements are significantly disjoint from the expert suggestions (e.g. request for *GeneralSolver* is made for a *SymmetricMatrix*), he will be provided with alternative suggestion(s) and thus may modify his request. Second, when user requirements are not very detailed, they will be made more specific by accommodating experts opinions. For instance, when user specified the problem, the matrix type (and size), and the algorithm, the system can additionally suggest the CPU / GPU type, and/or memory, and/or number of processors. Similarly, when the user specified only the problem and the matrix type, the expert knowledge and the AHP shall be utilized to suggest the algorithm and resources to be used.

In the *AiGExpertOntology* we follow, earlier specified, guidelines for ontology engineering, e.g. naming conventions for classes and properties, filling annotations for ontology elements. Moreover, we decided that new ontology has to become a new module (separated from previously designed ones).

5 Concluding remarks

The aim of this paper was, first, to discuss important issues involved in ontology reengineering, based on our experiences with the *AiG* ontology. Here, we have discussed problems that one can encounter when ontology has been created using earlier state of the art knowledge and tools and has to be extended and modernized. Second, to introduce a new ontology that is going to be used in the user decision support in the *AiG* system. This ontology has been developed following the

guidelines established during the reengineering process. The reengineered ontology is available at: `http://gridagents.sourceforge.net/AiGGridOntology`. Our current goal is to continue development of the ontology of computational linear algebra and apply it in a prototype of the user decision support subsystem.

Acknowledgments

Work of the authors was in part supported by bilateral grant between Polish Academy of Sciences and Bulgarian Academy of Sciences.

References

1. OWL 2 Web Ontology Language. `http://www.w3.org/TR/owl2-overview/`.
2. Fensel Dieter. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, New York, 2003.
3. 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.
4. 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.
5. 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.
6. 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.
7. M. Lucks. *A Knowledge-Based Framework for the Selection of Mathematical Software*. PhD thesis, Southern Methodist University, 1990.
8. 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.
9. 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.
10. Katarzyna Wasielewska and Maria Ganzha. Using analytic hierarchy process approach in ontological multicriterial decision making – preliminary considerations. In *Proceedings of 4th International Conference–AMiTaNS'12 Memorial Volume devoted to Prof. Christo I. Christov*, pages 95–103, 2012.