*Appeared in: PDP 2011, http://dx.doi.org/10.1109/PDP.2011.24, IEEE*

# Job Scheduling with License Reservation: A Semantic Approach

Jorge Ejarque*‡, Andras Micsik§, Raül Sirvent*, Peter Pallinger§, Laszlo Kovacs§ and Rosa M. Badia*†
*Grid Computing and Clusters Group - Barcelona Supercomputing Center (BSC), Barcelona, Spain
† Artificial Intelligence Research Institute - Spanish National Research Council (IIIA-CSIC), Barcelona, Spain
‡ Computer Architecture Department - Universitat Politecnica de Catalunya (UPC), Barcelona, Spain
§Distributed Systems Department - Computer and Automation Research Institute
Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
{jorge.ejarque, raul.sirvent, rosa.m.badia}@bsc.es, {micsik, pallinger, laszlo.kovacs}@sztaki.hu

***Abstract*—The license management is one of the main concerns when Independent Software Vendors (ISV) try to distribute their software in computing platforms such as Clouds. They want to be sure that customers use their software according to their license terms. The work presented in this paper tries to solve part of this problem extending a semantic resource allocation approach for supporting the scheduling of job taking into account software licenses. This approach defines the licenses as another type of computational resource which is available in the system and must be allocated to the different jobs requested by the users. License terms are modeled as resource properties, which describe the license constraints. A resource ontology has been extended in order to model the relations between customers, providers, jobs, resources and licenses in detail and make them machine processable. The license scheduling has been introduced in a semantic resource allocation process by providing a set of rules, which evaluate the semantic license terms during the job scheduling.**

***Keywords*-multi-agent, semantics, scheduling, resource allocation, software licenses, grid computing, cloud computing, distributed systems.**

## I. Introduction

The Cloud Computing [1] paradigm aims to offering computational resources (infrastructure, platform and software) as services across the Internet. In cloud computing, different applications belonging to different Independent Software Vendors (ISV) are executed by several users and Service Providers (SP) on resources owned by different Infrastructure Providers. In such a scenario, the management of software licenses is a crucial issue.

ISVs distribute their software under licenses with a wide variety of limitations and restrictions, which must be respected when the software is executed in the cloud. On the other side, Service and Infrastructure Providers have to be licensed to execute the required software requested by the users, so they also have to understand the license term and they have to check if their users are allowed to execute the requested software. However, a software license is typically a legal text with a wide variety of terms, which can not be automatically processed by machines. Therefore, the current systems for allocating licenses are bound to a set of license types which support few license terms. Moreover, the extension of these systems for supporting new license types and terms requires a lot of effort, because system developers have to implement the functionality for understanding and managing each type of license and term.

Based on these assumptions, we propose a semantic approach for scheduling the jobs in Service and Infrastructure Providers taking into account the different software licenses. In this paper, we suggest a way to sematically describe software licenses as resources and license terms as resource properties. In this way, computers can understand license terms and consequently, licenses can be automatically allocated to the different requested jobs as another kind of resource according to their properties (license terms).

The paper is distributed as the following: Section II describes some related work, which is interesting for our approach; Section III presents our semantic model for the different types of software licenses as resources; afterwards, Section IV shows the software licenses in the semantic resource allocation mechanism; and Section V presents a usage scenario used to validate the license scheduling including some measurements and implementation details. Finally, Section VI concludes the paper and gives some guidelines for future work.

## II. Related Work

There is quite few work done in the area of semantic modeling of software licenses and even fewer attempts to use such models in the management of service-oriented infrastructure.

In [2] the authors provide an ontological framework for the description of software license agreements. Their ongoing work is in the direction to facilitate autonomic license management with the help of knowledge management tools. It is not known in the literature whether this effort successfully reached its goals. The paper [3] tackles a similar problem in the area of Digital Rights Management (DRM). There are several languages for rights expression, which are non-interoperable with each other. The authors suggest a generic model for DRM from which several specific rights expression formats can be generated. The Creative Commons Rights Expression Language (ccREL) [4]

provides ways to express license properties in RDF [5]. License properties are categorized with keywords such as permits, prohibits, requires, etc. The possible values for permits includes reproduction, distribution and derivative works, which shows that the ontology is focused on media and documents, and therefore cannot be used as it is for software licenses.

Regarding overall issues about software licenses we would like to highlight the SmartLM project [6]. It is a research project for introducing the license management in distributed platforms such as grids and clouds. The approach of this project is to offer the licenses as services and negotiate the usage by means of Service Level Agreements (SLA). Despite of the differences on modeling between our approach (licenses as services instead of resources), we feel that some of the results of our work can be interesting for this project.

## III. Software License Ontology

Software licenses are key components of distributed processing, especially the planning of the execution of long-running, computation-intensive workflows often used in Grid [7] and Cloud environments. The lack of enough licenses can block the execution of a workflow similarly to the lack of appropriate computing resources. Despite that, the topic of intelligent license allocation is rather undeveloped today. One cause may be that licenses are a type of legal texts, and the machine understanding or machine understandable representation of such texts is problematic. Furthermore, there exists a huge variety of licenses, and the complete conceptualization of software licenses seems almost impossible. Hence, we try to take a practical approach; firstly, we only work with a subset of license content minimally required for job scheduling, secondly, we provide a pragmatic but extensible core for the semantic description of software licenses, and thirdly we limit the semantic descriptions to the viewpoint of resource management.

In the sense of the above criteria, software licenses are described with the following key properties:

- The licenser (the provider of the license)
- The licensed software
- The license terms containing the conditions for using the software

The great variety of license terms can be grouped into some main categories. The first category is about *time*: the software may be run in given daily periods, the license may have an expiration, or the license may be consumptive (a fixed number of runs is allowed), etc. Our second category is about the *location*: in which countries, in which places, on which IP addresses the software can be used. Often, the license is restricted to a single Grid or Cloud or to a single

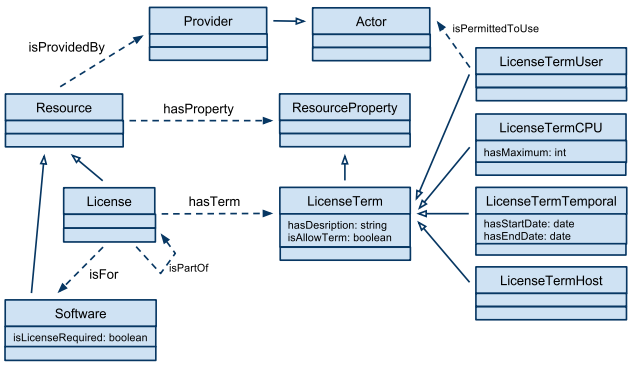


Figure 1. Excerpt of classes and relations in the resource description ontology

host. Another subtype of this class is the export restrictions to certain countries. The third category is about the *customer*; who can execute the software? This is somewhat related to the second category, as the country, the employer, or the legal status of the customer may be involved. Typically, software is provided to legal entities (such as companies, institutes, etc.) or individuals. Furthermore, re-selling of the software in terms of Grid-like computations for third parties may be regulated as well. The fourth category defines the level of allowed *parallel executions*; how many instances of the software may be run simultaneously? Often, the license is broken up into base licensing and additional, per-CPU licensing.

Our contribution in this paper is the extension of a current scheduling and monitoring environment for job executions [8] with license management. The existing environment was based on an ontology describing the most important concepts for our task. The ontology is derived from the Grid Resource Ontology (GRO) [9], which conceptualized the Grid domain as the result of several EU projects, and contains more than 100 concepts and relationships. GRO includes the concepts for processes, infrastructure, users, security, tasks and jobs. GRO is capable for the semantic description of customer jobs and resources but it misses concepts for describing resource allocation and historical data. The GRO has been extended in the scope of the BREIN project [10] in order to cover these aspects, and was further enriched towards the description of software licenses as depicted in Figure 1.

Within our ontology, resource requirements have been introduced as a set of required abstract *GRO Resources* in the *GRO Task* definition, which models an abstract job. This definition has been also extended with time constraints such as expected duration, deadline, earliest possible start, etc. The scheduling result (assigned resources and time slot) has been introduced on the *GRO Process*, which incarnates the *GRO Task*. *GRO Process* and *Task* descriptions are also

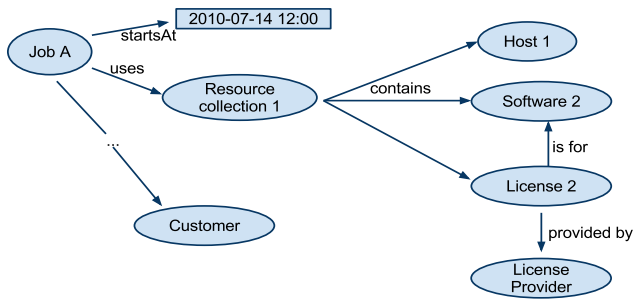


Figure 2. Example for a scheduled job and its allocated resources including a license

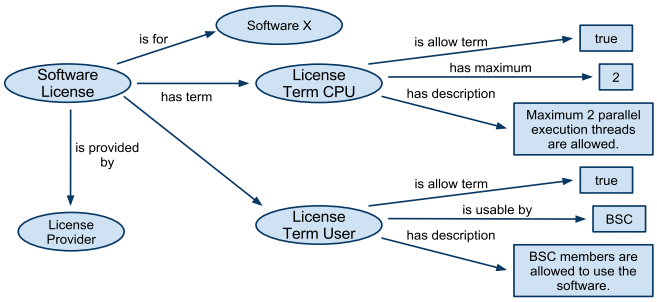


Figure 3. Example for a software license description

useful for recording past events and the schedules for future events.

Required resources are linked to the task definition as resource sets. A resource set is typically a host, a container of more detailed resource descriptions (disk, CPU, etc.) with resource properties (e.g. size of disk). Furthermore, tasks are also linked to their representing agents and to related business information (such as customer data, service provider, agreed SLA). Furthermore, the job description for the task contains the concrete plans of the service provider to fulfill the task requirements, and thus it contains all selected scheduling parameters and resources (Figure 2).

Our aim was to further extend task requirements and job descriptions to cover the requirements about software licenses. A software license is a consumable resource for which the capacities and consumption has to be planned and managed likewise usual Cloud or Grid resources. Therefore, it was a logical step to attach license instances as required and/or allocated resources to tasks. Thus, a scheduled job contains a set of resources to be used and this resource set contains the used licenses as well. A further advantage of this approach is that the *SoftwareLicense* subclass of the *Resource* class, by its parent properties, is linked to the business point of view of the executing environment, so for example the issuer of the license can be semantically described as a special type of provider. Furthermore, the use of the licenses can be recorded as resource usage entities similarly to the scheduled usage of hosts, etc.

The actual content of the software license is reflected in the license terms (Figure 3). Each *LicenseTerm* describes one type of restriction or permission contained in the license. For example, a license term may restrict the use of the licensed software to a single IP address or to a single CPU. License terms may be permissive or restrictive, and furthermore they can be categorized based on the target of the term. Based on our use cases we defined some basic categories for license term as a starting point based on their targets:

- **CPU:** is about simultaneous use of the software on multiple processors.
- **Host:** is about the set of hosts where the software may be run.
- **Temporal:** is about intervals and frequencies the software may be run.
- **User:** is about the possible users of the software.

The complete formalization of license terms would have gone far beyond the scope of our effort, therefore we provided a hybrid representation of license terms in OWL [11]; the license term individual contains the original text from the license, and additional properties manually extracted from the text. Term properties can be modelled and extended on demand based on the current application scenario. In our case the maximum number of CPUs, the expiry date of the license, the list of allowed hosts and users were explicitly defined in the ontological description of licenses.

A complete model of licenses in fact may contain arbitrary boolean combinations of license terms, i.e. expressions created using atomic terms, AND, OR, and parentheses. The presented solution is a simplification, which makes implementation feasible and fits most of our use cases and applied software licenses. Our license terms for one license are in AND relationship, which means that all terms of the license must be satisfied. The OR relationship can be modeled by providing several ontology instances for a single license, of which only one has to be satisfied. These alternative instances of the same license are grouped together (using the *isPartOf* property) so that they cannot be used simultaneously for different jobs.

The more complex cases for checking license applicability can be handled with rules. Rules can be used to place flags of incompatibility into the description of a scheduled job, thus preventing the selection of certain resource-license-user combinations. The set of such rules can be extended on-demand, as it is required by the software options offered by the service provider. In practice, we checked two alternatives to implement such rules: SWRL [12] and Jena built-in rules [13], and selected Jena built-in rules as they provide more flexibility and comfort in the specific

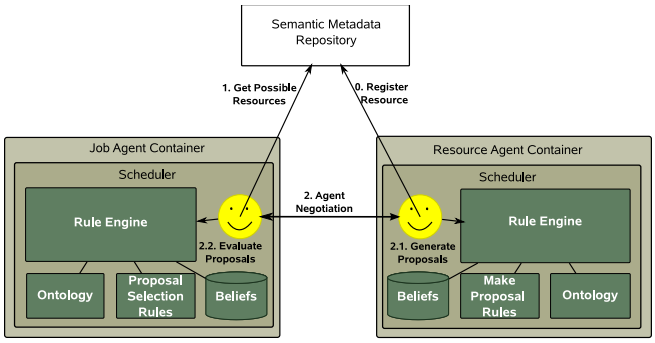


Figure 4. Sematic Scheduling framework overview

implementation. For example, such rules can check if the customer is affected by any export restrictions, whether the scheduled execution is after the expiry of the license, etc. Specific examples for these rules are given in next section.

## IV. Semantic Scheduling of Licenses for Jobs

In our previous work, a framework has been developed for allocating resources to the jobs and service executions requested by different users [8]. This framework combines different promising technologies such as multi-agents for improving adaptability and distributed problem solving and semantics for improving the extensibility and interoperability with heterogeneous resources from different providers. Due to the semantic capabilities of the framework, we opted for the introduction of semantic scheduling of licenses as another kind of resource which must be allocated according to the job requirements.

Figure 4 shows an overview of this framework which is composed by four different components. There are two differentiated parts in the architecture: the part which is related to the management of jobs and the part which is in charge of resource provisioning. The job management part allows customers to make all actions related to their jobs (submit, cancel,etc.), while the resource provisioning part allows the system administrator to add and remove resources.

Both parts are built on top of a JADE agent platform [14]. The platform can be distributed across multiple locations deploying containers on each of them. Moreover, it implements a messaging system, which allows the communication between agents located on different containers. The distributed configuration of the agent platform can improve the scalability of the system because the different parts can be processed in parallel on multiple hosts.

Customers' jobs and resources are represented in the system by software agents. Job Agents (JA) are in charge of managing the customers' jobs and Resource Agents (RA) are in charge of managing the providers' resources. Moreover, the scheduling of jobs in the different resources is made by an agreement reached from a negotiation between a Job Agent and different Resource Agents. Apart from the job management and resource provisioning parts, the system architecture contains a Semantic Metadata Repository (SMR).

During the initial setup (Figure 4 interaction 0), the RAs register the description of their managed resources into the SMR to enable the resource for job executions. The resource description includes the resource properties, the available software and their licenses. When a job is submitted to the system the job request is semantically annotated identifying the customer who has requested the job and specifying the resource requirements (including the software and type of license) and time constraints. Afterwards, the framework builds a semantic query based on the job description in order to find the capable resources for executing the job (Figure 4 interaction 1). Following lines show an example of this type of semantic query.

```
SELECT DISTINCT ?id ?host ?license
WHERE {
  ?host tech:isProvidedBy ?rm .
  ?rm  biz:actor_name ?id .
  ?host tech:containsResource ?proc .
  ?proc rdf:type gro:Processing .
  ?proc gro:hasResourceProperty ?memory .
  ?memory rdf:type tech:MemoryCapacity .
  ?memory tech:Mbytes ?mem_cap .
  FILTER (?mem_cap >= 1024^^xsd:int) .

  ... more filters ...

  ?host tech:containsResource ?image .
  ?image rdf:type tech:Image .
  ?image tech:containsResource gro:Software_XX .
  ?license rdf:type gro:License .
  ?license license:isForSoftware gro:Software_XX }
```

The first part of the query looks for computing resources whose properties match with the properties described in the resource requirements. The final part of the query checks if the software is installed in the available images for the computing resource and gets the available licenses for this software. All the capable resource combinations selected by the query are linked to the job request description using the *hasCandidateResources* property as shown in the next example. *HostA* and *lic_softX* are one of the candidate resource combinations selected by the SPARQL [15] query for fullfilling required host and license software.

```
<rdf:Description rdf:about="gro:job1">
  <rdf:type rdf:resource="gro:Execute_Task"/>
  <gro:hasActionState rdf:resource="tech:requested"/>
  <gro:hasResourceSet rdf:resource=gro:Requirement_job1"/>
  <tech:hasCandidateResources rdf:resource= "gro:job1_SetA"/>
  <tech:hasCandidateResources rdf:resource= "gro:job1_SetB"/>
  <tech:hasCandidateResources rdf:resource= "gro:job1_SetC"/>
  <tech:hasDeadline>2010-08-01T00:40:00</tech:hasDeadline>
  <tech:hasEarliestStartDate>2010-08-01T00:00:00
  </tech:hasEarliestStartDate>
  <tech:hasLatestStartDate>2010-08-01T00:20:00
  </tech:hasLatestStartDate>
  <tech:expectedDuration>20</tech:expectedDuration>
</rdf:Description>
```

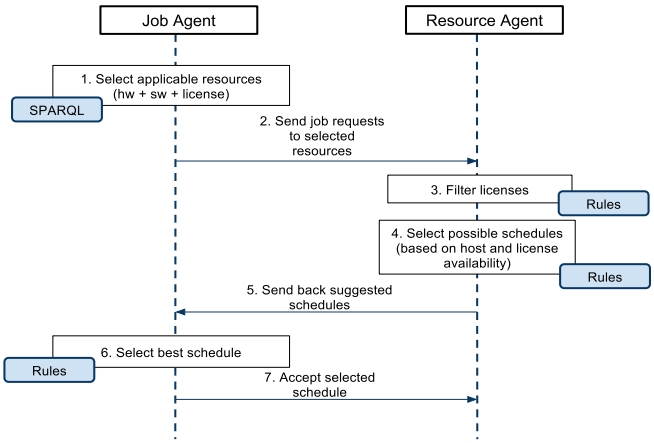


Figure 5. Semantic scheduling negotiation sequence

```
<!--requirements-->
<rdf:Description rdf:about="gro:Requirement_job1">
  <rdf:type rdf:resource="gro:Resource_Set"/>
  <gro:isPartOfExecuteTask rdf:resource="gro:job1"/>
  <gro:containsResources rdf:resource="gro:Req_job1_host"/>
  <gro:containsResources rdf:resource="gro:Req_job1_licX"/>
</rdf:Description>
...
<!--selected resources-->
<rdf:Description rdf:about="gro:job1_SetA">
  <rdf:type rdf:resource="gro:Resource_Set"/>
  <gro:containsResources tech:host_A"/>
  <gro:containsResources license:lic_softX"/>
</rdf:Description>
...
```

After the selection of candidate resources and their Resource Agents, the Job Agent initiates a negotiation with the selected Resource Agents in order to find the best allocation for the job (Figure 4 interaction 2). A job allocation in our system is a job incarnation which uses candidate resources during a time slot.

Figure 5 shows sequence for deciding the job allocation according to semantic descriptions. Once the Job Agent has identified the Resource Agent, which contains the candidate resources (Step 1), it sends a call for scheduling proposals to the selected RAs including the job description with the selected candidate resources (Step 2). Each Resource Agent will evaluate the proposal according to the resource provider scheduling rules taking into account the requested job description, the resource description and the current resource load (jobs assigned to its resources).

The provider scheduling rules can be separated in two types the scheduling part, filtering rules (Figure 5 Step 3), which examines if the selected fulfills the license terms, and the scheduling rules, which discovers time slots for allocating the jobs (Figure 5 Step 4). The filtering rules examine each applicable license in turn, and if they find any license term which forbids the use of the software, the currently examined resource collection is discarded from

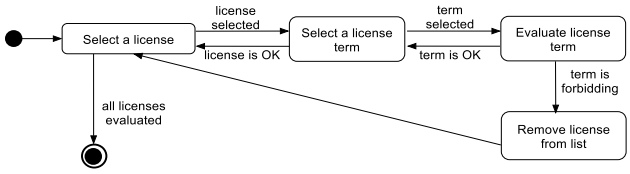


Figure 6. Filtering evaluation sequence

the candidate list. The rules in charge for checking specific types of license terms are pluggable, and therefore they can be dynamically adapted to the currently available collection of licenses (Figure 6). Next lines show examples of rules for filtering licenses.

```
[evaluate_license_terms_host:
    (?job rdf:type gro:Execute_Task),
    (?job gro:hasActionState tech:requested),
    (?job gro:incarnatedToProcess ?incarnation),
    (?incarnation gro:usesResource ?candidate),
    (?candidate gro:containsResources ?resource),
    (?resource rdf:type tech:Host),
    (?candidate gro:containsResources ?license),
    (?license rdf:type license:SoftwareLicense),
    (?license gro:hasResourceProperty ?license_term),
    (?license_term rdf:type license:LicenseTermHost),
    (?license_term license:isAllowTerm ?allow),
    equal(?allow, 'true'),
    noValue(?license_term license:hasTargetHost ?resource)
    ->
    print('Deleting incarnation because of host terms'),
    drop(2)]
```

The scheduling part (Figure 5 Step 4) is in charge of discovering the available time slots in the candidate resources where the load is not too high for executing the job and fulfills the license terms, time constraints and other provider rules. This task requires some procedural programming (e.g. Java) for the calculation of non-overlapping time slots for jobs, which is enhanced with logical rules to express preferences between pre-calculated schedules.

After the evaluation of the scheduling, Resource Agents send the scheduling proposals to the Job Agent (Figure 5 Step 5). At this moment, the Job Agent has all possible resource allocations (selected resources and time slots) which fulfill the time constraints as well as hardware and software requirements including the license terms. In the final step of the negotiation (Figure 5 Step 6), the Job Agent selects the best option according to the customer selection rules. Following lines give an example of a simple selection rule for selecting the allocation with the earliest planned start date. This rule eliminates an incarnation if there is another incarnation for this job which starts earlier.

```
[Earliest_startDate:
    (?job rdf:type gro:Execute_Task),
    (?job gro:hasActionState tech:requested),
    (?job gro:incarnatedToProcess ?incarnationA),
    (?incarnationA rdf:type gro:ProcessInstance),
    (?incarnationA gro:hasPlannedStart ?stDateA),
```

```
(?job gro:incarnatedToProcess ?incarnationB),
(?incarnationB rdf:type gro:ProcessInstance),
(?incarnationB gro:hasPlannedStart ?stDateB),
lessEqual(?stDateA, ?stDateB),
->
remove(5),
print('Incarnation eliminated because there
is another earlier')]
```

## V. Implementation and Validation

The semantic resource allocation framework components have been implemented in Java. The different agents have been developed with a BDI model [16] implemented by the Jadex engine [17] on top of a JADE agent platform [14]. The agent negotiation for allocating resources has been implemented using the Contract Net Protocol (CNP) [18] and the OWL descriptions for resources and licenses as well as queries and rules are handled by the Jena Semantic Web toolkit [13].

In order to validate the concept of semantic scheduling of licenses for jobs, we have set up a test environment. It consists of a submission of a set of jobs with different license requirements into a Service Provider which is licensed to execute different software under different terms. The selected terms for doing the validation are the Host, User, CPU and temporal terms. These terms for the sample set of software are summarized in Table I.

| License | Licensed Software | Terms |
|---|---|---|
| Lic_SwX | Software X | Allowed Hosts: HostA, HostC<br>Expiration: 2010/08/01 00:21 |
| Lic_SwY | Software Y | Allowed Users: User1, User2 |
| Lic_SwZ | Software Z | Maximum CPU: 2 |

Table I
Available Software Licenses

The SP contains 3 hosts (A,B,C) from 3 different resource providers with the same characteristics; Intel dual core processor 3GHz with 1GB RAM with an installed image which contains all the licensed software.

The Resource Agent contains the scheduling rules for evaluating the terms deadline, load on hardware resources and licenses with user, host, temporal and CPU terms. On the other hand, the Job Agent contains the selection rules which establishes a priority policy between the different resource providers and an earliest start date policy between incarnations proposed by the same provider. The rule below eliminates all but the earliest incarnation from each service provider:

```
[Earlier_startDate_same_provider:
    (?job rdf:type gro:Execute_Task),
    (?job gro:hasActionState tech:requested),
    (?job gro:incarnatedToProcess ?incarnationA),
    (?incarnationA gro:hasPlannedStart ?stDateA),
    (?incarnationA gro:usesResource ?candidateA),
    (?candidateA gro:containsResources ?resourceA),
    (?resourceA tech:isProvidedBy ?provider),
    (?job gro:incarnatedToProcess ?incarnationB),
    (?incarnationB gro:hasPlannedStart ?stDateB),
    (?incarnationB gro:usesResource ?candidateB),
    (?candidateB gro:containsResources ?resourceB),
    (?resourceB tech:isProvidedBy ?provider),
    lessThan(?stDateA, ?stDateB)
    ->
    remove(7),
    print('Incarnation which starts', ?stDateB ,
    'eliminated because there is another earlier',
    ?stDateA)]
```

A priority can also be established between resource providers. For example, if we prefer provider A to provider B, we can formulate it as the following rule:

```
[Priority_AtoB:
    (?job rdf:type gro:Execute_Task),
    (?job gro:hasActionState tech:requested),
    (?job gro:incarnatedToProcess ?incarnationA),
    (?incarnationA gro:usesResource ?candidateA),
    (?candidateA gro:containsResources ?resourceA),
    (?resourceA tech:isProvidedBy biz:ResourceProviderA),
    (?job gro:incarnatedToProcess ?incarnationB),
    (?incarnationB gro:usesResource ?candidateB),
    (?candidateB gro:containsResources ?resourceB),
    (?resourceB tech:isProvidedBy biz:ResourceProviderB)
    ->
    remove(6),
    print('Incarnation from providerB eliminated
     because there is another from providerA')]
```

We have submitted a set of jobs for validating the behavior of the system focusing on the allocation of licenses to jobs according to the different license terms. Table II shows the requirements for each job and the scheduling results. In terms of hardware resources and time constraints all jobs consume the whole machine during the same duration (20 min) and have the same deadline (2010/08/01 00:40:01).

The first two jobs are allocated in *HostA* because its provider has more priority than the others. *Job 3* and the next ones cannot be allocated later on *HostA* because this allocation exceeds the job deadlines, so *Job 3* is allocated on *HostB* consuming 2 CPU during the allocated interval. According to the user term of the license for *Software Y*, there is no possible allocation for *Job 4* because the customer who requested the job (*User3*) in the allowed list. *Job 5* fullfils the user term because it is requested by *User2*, however it has been allocated in *HostC* because *HostB* is not an allowed host for executing *Software X*. *Job 6* could not be allocated because of the possible allocation in *HostC* exceeds the temporal term of license for *Software X*. Finally, *Job 7* was allocated to *HostB* as it satisfies all license terms.

We have measured the reasoning time of Resource Agents and Job Agents. This time depends on the complexity of the rules specially when they use built-ins and the quantity

| ID | Cust. | Required Software | Planned Start Date | Planned End Date | Assigned Resources |
|---|---|---|---|---|---|
| 1 | User1 | Sw X | 2010/08/01 00:00:00 | 2010/08/01 00:20:00 | HostA Lic_SwX |
| 2 | User2 | Sw Y | 2010/08/01 00:20:00 | 2010/08/01 00:40:00 | HostA Lic_SwY |
| 3 | User3 | Sw Z | 2010/08/01 00:00:00 | 2010/08/01 00:20:00 | HostB Lic_SwZ |
| 4 | User3 | Sw Y | - | - | None-User3 not allowed |
| 5 | User2 | Sw X Sw Y | 2010/08/01 00:00:00 | 2010/08/01 00:20:00 | HostC Lic_SwX Lic_SwY |
| 6 | User3 | Sw X Sw Z | - | - | None-Lic. expiration |
| 7 | User1 | Sw Y Sw Z | 2010/08/01 00:20:00 | 2010/08/01 00:40:00 | HostB Lic_SwY Lic_SwZ |

Table II
JOB SCHEDULING ACCORDING LICENSE TERMS

of descriptions to evaluate. For this simple validation test the Resource Agent takes about 0.5 seconds to generate the allocation proposals. On the other hand, the Job Agent takes less than 0.2 seconds for selecting them. These times can be considered reasonable compared to the execution times of the jobs executed in platforms such as Clouds.

## VI. CONCLUSION

The paper has presented a semantic approach for scheduling jobs taking into account the issues with software license allocation. This approach defines the licenses as resources which can be allocated for jobs according to their terms. A resource ontology has been extended in order to model the relationships among resources, licenses and license terms. The license scheduling has been introduced in a semantic resource allocation process exploiting RDF querying and custom rules based inferencing.

This framework is the first working implementation known to us that supports the scheduling of license usage based on semantic descriptions. Furthermore, it is a flexible and extensible solution, where issues raised by new types of licenses can be covered by new rules plugged into the running environment. This lightweight approach tries to keep reasoning tasks at a low level to ensure the speed of inferencing, yet our validation tests approve that it is sufficient for helping to manage the license allocations appearing in usual high performance computing scenarios. The legal terms within the licenses cannot be modeled fully in our case, but we think that we managed to keep the balance between the expressivity of the semantic license descriptions and the feasibility and efficiency of the implementation.

Our future work is focused on the development of new rules for modeling more complex license types as well as more complex policies for scheduling licenses.

## ACKNOWLEDGMENT

This work is supported by the Ministry of Science and Technology of Spain and the European Union under contract TIN2007-60625 (FEDER funds), Generalitat de Catalunya under contract 2009-SGR-980 and the European Commission with FP6-IST project 34556 and FP7-ICT project 215483 (S-CUBE).

## REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[2] Q. Zhao and M. Perry, "An ontology for autonomic license management," in *4th Int. Conference on Autonomic and Autonomous Systems,(ICAS 2008)*, 2008.

[3] Nadah, N., de Rosnay, M. D., and Bachimont, B., "Licensing digital content with a generic ontology: escaping from the jungle of rights expression languages," in *11th international Conference on Artificial intelligence and Law*, 2007.

[4] H. Abelson, B. Adida, M. Linksvayer, and N. Yergler, "ccREL: The Creative Commons Rights Expression Language version 1.0," 2008.

[5] "Resource Description Framework," http://www.w3.org/RDF.

[6] "EU SmartLM project," http://www.smartlm.eu.

[7] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 2004.

[8] J. Ejarque, R. Sirvent, and R. M. Badia, "A multi-agent approach for semantic resource allocation," in *The 2nd International Conference on Cloud Computing Technology and Science*, 2010.

[9] "Grid Resource Ontology," http://www.unigrids.org/.

[10] BREIN Consortium, "Final Report on the BREIN Core Ontologies," BREIN Project, Public Deliverable D3.2.5, 2008.

[11] "Ontology Web Language (OWL)," http://www.w3.org/TR/owl-features.

[12] "Semantic Web Rule Language," http://www.w3.org/Submission/SWRL.

[13] "Jena Semantic Web Framework," http://jena.sourceforge.net/.

[14] "Java Agent DEvelopment Framework," http://jade.tilab.com/.

[15] "SPARQL Query Language for RDF," http://www.w3.org/TR/rdf-sparql-query.

[16] A. Rao and M. Georgeff, "BDI agents: From theory to practice," in *The 1st Int. Conf. on Multi-agent Systems*, 1995.

[17] "Jadex system," http://jadex.informatik.uni-hamburg.de/.

[18] R. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. on Computers*, vol. 100, no. 29, pp. 1104–1113, 1980.