Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems

Isaac Lera*, Carlos Juiz, Ramon Puigjaner

Universitat de les Illes Balears, Ctra. de Valldemossa, km. 7,5, 07071 Palma, Spain

Received 3 March 2005; received in revised form 19 September 2005; accepted 8 November 2005

Abstract

Several techniques and applications have been proposed to aid the decision taking process in the system performance domain. Most of these techniques have depicted the performance model of systems through annotations of performance measurements coming from specific software descriptive syntactical languages. However, the semantic representation of performance information provides the possibility of its ulterior machine-processable logical interpretation and therefore the applicability of inference rules about a particular domain. Moreover, ontologies ease the interchange and reuse of knowledge of particular domains, e.g. system performance. In this work, we propose a performance ontology together with the system performance analysis technique as an example of framework building for intelligent applications based on semantic web. The paper also shows the construction of performance rules through OWL to automatically infer new performance constraints and QoS knowledge about the system on execution.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Performance engineering; Semantic web; Ontology engineering

1. Introduction

This paper shows that it is possible to reason about the performance activity in an ambient intelligent environment and even to take action on it during execution. Thus, the huge amount of knowledge that was gathered under the Software Performance Engineering (SPE) discipline has to embrace new issues related to ontologies. The conceptual framework of ambient intelligent systems includes a context of the execution that structures the adaptability of the system to the users. The context of the execution is an information space that can be modelled may satisfy, as a set of entities (objects), a set of rules (functions) that entities may satisfy a set of relations among the entities and a set of situations, therefore the model has an ontological foundation. An ontology is a formal, explicit specification of a shared conceptualization.

The construction of ontologies relies on previous models in order to build a shared domain of knowledge with semantic richness. The importance of using an ontology as a conceptualization formalism resides on the potential power of reasoning over the represented data. The ability of reasoning will allow taking decisions about performance
and other non-functional features of systems. In particular, intelligent agents may be executed on different scenarios sharing the semantic representation of performance values.

Thus, the motivation of this work is determined by the almost complete absence of techniques in the ontology engineering field to support an intelligent analysis of system performance. In other performance techniques, most of the tools use their own language either to annotate measurements or to interchange formats among modelling and evaluation tools. Reuse of measurements is done by structural relationships (XML parsers or relational data bases) but semantics are not considered. However, there are new areas where the semantic richness of information can help in the process of decision making. For example, in information systems, searching functions for personnel contracts and other business data are chaotic and task consuming. The use of semantic information techniques can ease these functions. In context-aware systems, where services depend on the specific situation of their actors, the intelligent semantic applications will also ease the interchange of performance knowledge, the performance prediction and the self-healing maintenance of systems.

This paper will be organized as it follows. In Section 2 we shall overview the construction of a conceptual model of performance. In the next section, we shall explain how to represent the performance domain through OWL, taking advantage of its features. In Section 4, an example of ontological application is built. This practical example illustrates how software agents could reuse the ontology. Section 5 will show, by example the inference rule construction. Finally, Section 6 summarizes the conclusions of this paper and provides an outlook to future work.

2. Performance knowledge modelling

Nowadays, the term ontology is becoming a new hot topic in different activities as Intelligence Artificial, Web Semantic and e-business. Google, eBay, Amazon, travel agencies and other enterprises boost the investigations and development in this way.

In 1993, Tom Gruber [5] defined the term ontology as an explicit specification of a conceptualization. It is referred to the shared understanding of some domains, which is often conceived as a set of entities, relations, axioms and instances [6].

There are different reasons for developing models based on ontologies:

- Knowledge sharing: It permits computational entities as agents and brokers to use a set of common objects on a specific context while they interact.
- Knowledge reuse: It builds an ontology integrating in its definitions other well-defined web ontologies of different knowledge (e.g. temporary, spatial, business, etc.).
- Logical inference: It uses different logic reasoning mechanisms to check the consistency of the ontology and the knowledge, to check for unintended relationships between classes and to automatically classify instances in classes.

Different languages have been defined to represent particular knowledge, e.g. DAML+OIL [3], but none was totally compatible with the semantic World Wide Web as the Web Ontology Language (OWL) [15]. OWL is built on Resource Description FrameWork (RDF) and RDF Schema, with a set of vocabularies for describing properties and classes [25]. OWL enables the definition of domain ontologies and sharing of domain vocabularies.

On the one hand, there are several issues to be considered when defining a framework for the performance assessment of intelligent applications. These seem to be similar to traditional software performance engineering techniques: (i) It must be decided the way the intelligent system is modelled, and therefore, how to add the performance-related information to the software specification with minimal interference; (ii) Once the performance aspects of the system are depicted in the model, it has to be defined how to transform the architectural options into performance models, and finally (iii) it may be considered the way for evaluating each performance model derived from previous steps. This typical SPE strategy does not consider the semantic representation of the information on the model.

On the other hand, an ontology is an explicit formal description of concepts in the domain. It is composed of classes, properties of each class, and restrictions on properties. Therefore, performance-related information may be also declared through this new web engineering approach, not only for the performance evaluation of the different components of the system, but also due to the intelligent application environment where it is possible to reason about the performance activity and even take actions based on this reasoning.

Different development paradigms have been lately used in Software Engineering (SE), e.g. entity-relationship, and more recently UML modelling. The software development process requires a set of concepts, a methodology and
its own language (formalism); this process is known as the software life-cycle which is basically composed by four stages: analysis, design, implementation and deployment [8]. The ontology development life-cycle is based on to the classical approximation in SE. We use the ontology building life-cycle depicted in Fig. 1, but it is explained more extensively in [2,22].

The following steps are normally used for building an ontology:

- Identify purpose and scope. Specifying the characteristic requirements is crucial to the design, evaluation and reuse of an ontology.
- Knowledge acquisition. What are the goals of the ontology? What kind of knowledge is needed?
- Languages and representation. Looking for languages that allow describing the domain and also providing ways to get the knowledge acquisition goals.
- Ontology building:
  - Conceptualization. To identify key concepts, their properties and the relationship among them in order to depict the knowledge domain.
  - Encoding, i.e. representing the conceptualization through a formal language.
  - Integration with existing ontologies.
- Available development tools, which supply building, documentation and evaluation tasks for the ontology.
- Evaluation, validation of corresponding initial goals and final results.

These methodologies together with OWL define our performance metamodel development. We concentrate our research on performance evaluation through operational analysis to dynamically compute interesting system parameters as the mean response time or even which is the bottleneck of certain applications. Performance values should allow taking decisions about performability, for example, setting the users’ system execution system or the required quality of service (QoS) for an application. All the decisions will be supported by a performance knowledge base.

As a first step, we will consider briefly the topic of our analysis: performance engineering. Which techniques exist and what parameters are necessary to represent it? Every performance evaluation study consists of two phases: the selection of an evaluation technique and selection of parameters. For the first phase, there are basically three main techniques: analytical, simulation and measurement. The parameter selection is determined by the representative objectives of our study (e.g., response time, demand, bottleneck detection,...). On this point, we can ask ourselves the following question: Does a performance ontology have to be able to embrace all these techniques and all the possible metrics? The answer depends on the domain amplitude. The ontology objective is to describe the particularity of a world, no matter the tools or techniques to operate it. For example, if we model a problem using simulation or queueing networks, we have always to develop other ontologies different than our proposal, where our basic elements are: resources, workloads, scenarios, etc., and their relations. In the same way, it is impossible to make a complete ontology that embraces all existing metrics for the selected parameters. Moreover, knowing the statistical distribution of inputs, that is crucial for queueing network analysis, this is not a requirement to apply simpler rules as the operational analysis laws.

Our performance metamodel is based on the UML Profile for Schedulability, Performance and Time specification by OMG [14,23]. An incomplete list of performance variables may be the following:

- Response time, defined as the interval of time between a request and the reply of a given transaction.
Throughput, defined as the rate at which the requests are serviced.

Utilization, defined as the probability of busy service or resource.

Demand, defined as the completion time of a composed set of requests.

And other variables such as: service time, latency, priority, capacity, observation time, etc.

Thus, these variables may be real positive numbers, probabilities, rates or even times but these syntactic descriptions are incomplete. The performance knowledge expressed through ontologies needs to answer questions such as the following: who are the sender and the receiver of a given transaction? which are the consequences of a system response time higher than the required QoS? which component is the bottleneck? how many transactions per unit of time support the bottleneck? etc. The answers of these questions require more information included in the performance model. Moreover, the agents that are collecting this data could be measuring, monitoring, estimating, simulating or even inferring the information to answer these questions (or giving clues about them).

In this work we distinguish four classes of values for performance variables: assumed, estimated, required and measured. These variables are representing object properties belonging to a specific domain. For example, the utilization of a processor could be monitored during task execution; therefore the ontology has to represent the measured object, i.e. the processor, as a resource, and the measured variable, i.e. the utilization, as a performance measure of this resource. Resources are active and passive, in a simple performance domain, the processor is an active resource since it is performing tasks (workload) instead of being accessed by others like the passive objects. Since the processor is active, it supports a workload which is expressed as the required demand for running in a specific scenario. This workload could be requested inside the system by other objects or coming from outside the model. This domain model is represented in Fig. 2 as the basic model.

In Fig. 2 (QoS Vision) the resource provides a service to agent. This service is acceptable if it complies with QoS Required by the agent and the parameters offered by the resource. For each resource of the system, its QoS requirements (and the specification QoS Offered) must be known.

However, our perception of the use relation is more intuitive and natural than this basic model. We may express the utilization, for example, as “the application X is using 50% of processor capacity”. On the other hand, the processor’s utilization could be understood as its use during a thread of execution or during the application run, which are completely different scenarios, shown in Fig. 2. Therefore, there are different scenarios in the same context with different granularity of representation. Each scenario is composed by the resources of interest; in particular the processor is represented by the host in the extension modelled in [12]. The model shown in Fig. 2 corresponds to the UML-SPT Profile [14,23] including a new QoS profile modification. In the next sections, we are going to develop this profile.

Building a Quality of Service (QoS) profile is motivated by practical reasons since the performance ontology could represent service level agreements (SLA) for intelligent systems. The functional behavior of a system component may
work properly but their output may be unacceptable from the QoS viewpoint. In that sense, intelligent systems require a decision process based on performance evaluation and the required QoS information. Therefore the model has to take into account special service features such as priorities, thresholds, etc. Following the example of the processor, the natural way to express its QoS could be “the processor could not exceed 90% of utilization during execution, if this occurs then…” and the system automatically triggers some predefined alarm rule to guarantee the SLA about a processor’s utilization [20].

3. Performance ontology building

The difficulty on defining a performance ontology for performance evaluation consists on the performance modelling of the system through the ontology language. This goal would require the ulterior offline evaluation of the system which would not be compatible with an intelligent performance assessment supported during the execution of the system. However, the objective of the online performance assessment of the intelligent system is to use the OWL description to store/analyze/reason the required information about the performance metrics of certain components while the system is running.

3.1. Overview of the OWL language and OWL tools

The OWL language is designed to be used by applications that need to process the content of information instead of only representing information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDFS) by providing additional vocabulary along with formal semantics. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL (short for Description Logic), and OWL Full. OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF [15] (see Fig. 3). The disadvantage of OWL Full is that the language has become so powerful as to be undecidable and inefficient to reasoning support.

OWL is based on object-oriented programming, where there are a set of domain facts: classes, relations and properties. Using OWL allows defining these concepts:

- Classes, as objects belonging to a specific domain.
- Attributes, defined in a local or global domain of a class.
- Facets or properties of attributes, restriction type, cardinality, etc.
- Taxonomies: subclasses can be defined, with multiple inheritances, as well as exhaustive and disjoint decompositions.
- Axioms: OWL allows the definition of first order logic axioms.
- Instances.

Working with OWL may require a text editor or other ontology tools such as KAON, Chimaera, WebOnto, KSL, SWOOP, etc. The Eclipse programming environment has a plug-in, namely SweDE to provide an extendable framework for integrating various Semantic Web tools. We select Kazuki, an object ontology generator, to be used together with Jena as we shall show in the next sections.
3.2. Representing performance objects and performance properties

Within our performance model we have defined a performance context where a set of scenarios are included. These scenarios are using a set of resources running a workload during some time period. For example, we define two workload classes (open and closed) in OWL, as Fig. 4 shows.

Since metrics are essential to describe performance, we define the resource utilization as a probability of being used. On the other hand, we also describe the work population as the number of jobs in a closed system. The probability and number classes are type values that the performance metrics adopt. These properties are shown in Fig. 5.

3.3. Reusing ontologies

One of the most interesting features of using OWL is sharing knowledge among different conceptualizations about domains. In this section we overview the way to import a time ontology and the QoS profile. Since we are working in the performance domain, the time ontology is crucial to operate with performance values and metrics. One of these performance metrics is the service time. The service time is a metric expressed with time units in a natural way. In other words, using simple data types to depict time units causes the performance ontology to lose reasoning power where it is combined with other ontologies [26]. Therefore, it is recommendable to reuse time ontologies, e.g. the DAML Time Ontology by the Information Science Institute that it may be imported as resource [7].

The construction of a QoS profile required by users could be a hard process. OWL-S [16] and supporting tools as agent technology enabling automation of services on the Semantic Web, ease the automation of Web service
tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring. OWL-S defines these ontologies:

– Service. Definition of the service, what is it doing? How does it work?
– Profile. It provides a definition of properties such as name of the service, contact information, quality of the service, and additional information that may help to evaluate the service, etc.
– Process. Description of the service operations as a process model. The control and the data flow are detailed here.
– Grounding. It depicts the service access mode.

Incorporating the QoS profile to a performance ontology completes the necessary knowledge about services to the intelligent agents requiring those services. In the same way that the knowledge about the requirements through user profiles is perceived, it is feasible to know the type of service that resources are offering. In Fig. 6, it is defined the maximum response time, the service parameters are collected in Pparameter to be instantiated by intelligent agents.

4. Working with a performance-related ontology

Once the OWL code has been implemented, it is necessary to build an external application (broker) for interpreting, computing and updating performance values on instantiations of clients and devices. Jena is a Java framework for building Semantic Web applications that support inference engines over OWL [9]. Coding classes, relations and properties could be done manually or through open source tools like Semantic Web Development Environment (SweDE). SweDE edits, visualizes and validates ontologies and offers a Java API named Kazuki [18]. Kazuki automatically builds Java instances based on Jena libraries.

Jena provides connectors to the main data base, e.g. MySQL, Oracle and PostgreSQL that could manage different brokers preserving the data consistency and coherence in concurrent accesses.

Thus, implementing an application to apply operational analysis over the performance ontology instances, is an approximation to the performance agent’s role. The knowledge domain of the agent is not only the set of performance parameters but also others that may belong to other ontologies through inheritance. Therefore, an intelligent system can be monitoring itself getting the values of these parameters and consequently compute others, e.g. through operational analysis, to feed back the instances of the performance ontology [10,11]. The system should be capable of making decisions from the knowledge (or the absence of knowledge) of the performance values together with other functional and non-functional activities. Some examples of applications that use the ontology knowledge could be the bottleneck detection or the QoS fulfillment.

5. Performance reasoning

The use of ontologies allows inferring new facts through the application of axioms and rules. In this section, we will describe context reasoning to show the importance of the use of ontologies to describe models. We divide this section in two categories: ontology reasoning based on description logic, and user-defined reasoning rules using first-order logic.
DL Reasoning Rules:


Explicit information:

```xml
<owl:ObjectProperty rdf:ID="isIn">
  <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</owl:ObjectProperty>

<Resource rdf:ID="Printer">
  <isIn rdf:resource="#Laboratory"/>
</Resource>

<Room rdf:ID="Laboratory">
  <isIn rdf:resource="#Office"/>
</Room>
```

Implicit information:

```xml
<Resource rdf:ID="Printer">
  <isIn rdf:resource="#Office"/>
</Resource>
```

Fig. 7. Reasoning using ontology.

5.1. Ontology reasoning

The equivalence of OWL and DL (Description Logic) allows OWL to exploit the existing DL reasoning. Description Logic is a knowledge representation formalism unifying and giving a logical basis to the well known traditions of frame-based systems, Semantic and Object-Oriented representations, etc. [24]

To illustrate this kind of reasoning we introduce a spatial ontology. This case is a simple example of spatial localization. Let’s assume that an office has a laboratory. This laboratory has a printer. Through a simple transitive property, we can determine that the printer is in the office. This simple property is shown in Fig. 7. We may extend the example considering a man in the office. Let’s assume that the man has been registered at the entrance but it is not possible to locate him. If this man is working with a computer then we can indirectly determine his position by the computer location and its use. This inference has been done through first-order logic. As we may observe it is difficult to reason about simple properties without considering first-order logic.

5.2. User defined-rules

The ontology reasoning is limited by the definition of the language. The user-defined rules, based in first-order logic, provide more flexibility. In this section, we provide an example about broker reasoning in order to approximate the selection of the task to be executed depending on the workload. Therefore, we define a complementary classification for the workload:

- Workload based on resource utilization. We define three disjoint classes corresponding to HardWork, MediumWork and LightWork. For example the Hard workload would be needed for image rendering, Medium for mp3playing and Light for showing the system calendar.
- Workload based on the priority of producer. We define two disjoint classes, namely, HighPriority and LowPriorityWork.
- Workload based on efficiency. We define two disjoint classes, corresponding to EfficientExecutionWork and Not EfficientExecutionWork.

We assume HardWork as that task which is using more than a threshold of resource utilization. In the case of changing the resource to another more powerful the task should abandon this classification. Therefore, the classification of tasks depends on the context of execution. In Fig. 8, the rule to define HardWork is specified.

Fig. 8. Rule definition HardWork.

Agent(?ag) ∧ WorkLoad(?wor) ∧ hasProducer(?ag, ?wor) ∧ priority(?ag, ?pr) ∧ swrlb:greaterThan(?pr, ?threshold) → HighPriorityWork(?wor)

Fig. 9. Rule definition of HighPriorityWork.

LightWork(?x) ∧ HighPriorityWork(?x) → EfficientExecutionWork(?x)

Fig. 10. Rule definition of EfficientExecutionWork.

We also use SWRL (Semantic Web Rule Language) from W3C, where rules are defined by implication of an antecedent and a consequent. The antecedent is composed by a set of atoms. The plugin of Protégé has an editor of rules that eases the syntactical construction with SWRL [19,13]. In Fig. 9, we show an example of a HighPriorityWork rule. Once we define the rules of the above we may define an efficient task as the one that uses the resource lightly but the work is produced by priority agents (see Fig. 9).

We may compare the rule in Fig. 10 to the OWL coded in Fig. 11. Both rule definitions seem to be equivalent; however there are several main differences. In the code of Fig. 11, the threshold has no influence on classifying the workload with the feasible variation of resource power. Moreover, in some cases a priority task could be light, in terms of work (e.g. an alarm controller) but may not be efficient in some other context. An external situation of the ambient could trigger (the alarm situation) other hard work tasks (alarm reply process) which were not efficient but take out the priority rights. In consequence, the rule definition may change the knowledge of a domain depending on the context where the ontology analysis is performed.
At QoS level, it is possible either to take actions or infer new knowledge from the rules’ application. For example, actions may stop the execution of a work when it does not comply with the QoS level. This action is shown in Fig. 12, where the quality parameter is the response time. On the other hand, it is possible to detect idle resources diminishing the overall efficiency of the application. An ambient intelligent broker would change the system configuration incrementing the workload of idle resources.

6. Application of the performance ontology

In order to illustrate the application of the performance ontologies, we are going to describe an intelligent environment application, Smart Meeting Room (SMR). SMR is an ambient intelligent environment application that provides useful meeting-related information to the people inside the meeting room where it is set up. These services include, not only the transference of lecture notes and updated schedules, but also a vision of who is in the room at the moment and the possibility to contact them directly to arrange a meeting. The goal is to have all attendants interconnected and well informed of the latest conference news.

All workplace-specific information is formalized in an ontology. People and relationships, schedules, services and resources are some of the customizable aspects. In this paper we are interested only in performance related information, that is, services and resources. Therefore, we need a description of the local services and the local resources. This description is made deriving the objects from the Performance Ontology. The CPU is an instance of a Processing Resource and a disk is an instance of a Passive Resource. This resource definition ontology is imported and read by the SMR in order to have an internal representation of the objects that can be measured to assess the performance.

In the same manner the services need to be defined and the information about their duration and their resource consumption has to be provided. This is done using a combination of OWL-S and our Performance Ontology.

During startup the SMR core reads these ontologies and creates an internal representation of the context. This representation is coded in the Jess language. SWRL does not include a reasoning engine so that it’s advisable to define rules in Jess format, both formats are similar syntax. Jess is a rule-based inference engine that our performance module uses to reason about the environment. Jess works with facts and rules that the performance module updates constantly with data from performance monitor tasks. This is simple to manage as there is a straightforward correspondence between any OWL ontology and the rules that Jess understands.

While the system generates this information context, the Jess engine checks the antecedent rules to verify its true. If any antecedent is truth then triggers its consequent. Active, Stop or Degree the service affects to the performance system.

As a final consideration, in Smart Meeting Room ontologies are used both in the customization phase, previous to operation, to describe the environment, and during execution to understand the events arriving from the environment, to reason about them and to take corrective actions in case of necessity. To recap, we proved that is possible to improve the perceived QoS of an ambient intelligent environment, as in the SMR example, through simple performance ontologies and well-defined rules.

7. Conclusions and future work

This paper addresses the use of OWL ontology definition as the solution to evaluate the performance and QoS of ambient intelligent systems. However, the first step is to show that the syntactic use of ontologies for performance evaluation may incorporate almost the information as annotated modelling languages in the SPE area.

Thus, we have built a modified version of the UML SPT profile including information to automatically infer QoS knowledge through heuristic laws. In order to implement several application examples we used the OWL language
from W3C together with Protégé software. We also have used some Java interfaces such as Jena and Kazuki. The import/export and integration facilities of ontology languages allow us to share promising extensions for services, such as OWL-S that we reused to define quality parameters.

Within the novelty of this work we may point out the possibility of inferring performance knowledge and the perceived quality of service during the execution of intelligent software agents. Future work has to consider other inference techniques and the reuse of other non-functional ontologies.

The expressiveness of the ontology language rarely substitutes the syntactic richness of other approaches for an off-line performance evaluation of the system. However, the semantics and the standardization of OWL persuades us to further research on this direction.

We listed the steps in the performance ontology-development process and addressed the issues of defining class hierarchies and properties of classes and instances. However, the ontology design is a creative process as modelling itself. Therefore, different ontologies designed by different people would be equivalent. The potential applications of the performance ontologies will affect ontology design choices by designers. This is a new feature inside the domain of performance evaluation of ambient intelligent systems.

References

[16] OWL-S supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services, http://www.daml.org/services/owl-s/.