

Web Performance and Behavior Ontology

Carlos Guerrero, Carlos Juiz, Ramon Puigjaner
Departament de Matemàtiques i Informàtica
Universitat de les Illes Balears
Crta. Valldemossa km. 7,5
07122 Illes Balears, (SPAIN)
{carlos.guerrero,cjuiz,putxi}@uib.es

Abstract

We present a web system architecture using ontologies to improve the behavior of the system from the performance viewpoint. Since web system performance indexes depend on state and parameter values on runtime period, the proposed system configuration will change during this period. In order to perform this change, the web system is monitored and gathered information stored into a knowledge base. We also model the performance of the different web system elements intervening in the configuration using the knowledge base expressed by means of ontologies. An example of the use of this ontology in cache tier is also presented. We propose the use of performance reasoners to change the configuration during runtime period based on the information supplied from the knowledge base.

1. Introduction

Web servers have completely changed the way their contents are delivered. One of the major changes introduced has been the traditional context generation in a dynamic way. At the beginning of web systems, contents were static and the server process to deliver that content only involved the task of reading a static file.

In newer web dynamic environments, the process to deliver contents to users incurs: 1) user requests are dispatched to appropriate software modules that service these requests, thereby producing network overhead; 2) these software modules determine which data to fetch and present, thereby producing process overhead; 3) the disk I/O management querying the back-end database produces data overhead, and finally; 4) the assembled data needs to be formatted and delivered to the browser, thereby producing cache overhead.

In short, the Web page on-the-fly building process involves a higher number of activities than in static web

sites. Therefore, the dynamic workload and its corresponding computational cost are also higher than the workload generated by a static web server. Thus, research topics to improve the performance of web systems are getting more importance [14, 15, 10].

Traditional web performance techniques are usually applied to specific parts of the web system. For the purpose of this process only the information gathered during web monitorization is used. In order to improve global performance of the web system, it is preferable to have data about all the system parts, and then take decisions to dynamically change the configuration of the system using that global information. However, we need to deal with the heterogeneity of the data coming from each system part. The tools and techniques used until now are not taking into consideration the global data. Instead of this, only a part of that data is considered acting over a restricted part of the system.

The aim of the research work here presented is to create a tool that changes the web system setup and configuration by using the global information of the system (users, servers, middle-tiers, performance, behavior, etc.). Recent research works present studies about the way that systems dynamically adapt themselves to the ambient by means of the the information about the state of the system and their environment [12, 13].

In order to consider the knowledge behind this information ontological languages are extensively used. Ontologies are the ideal way to manage and to analyze information from automatic resources. Data about the state of the environment and the performance of the system are stored by instances of an ontology in a homogeneous and formal way. The same idea is applied our work.

In our framework, ontologies manage the information about the state of the system (Web System Elements Knowledge Base) and the behavior of the user and the servers tiers (Behavior Knowledge Base). A reasoner analyzes the information provided by the Behavior K.B. and applies operations to the instances of the different Web System Elements

K.B.

The remainder of the paper is organized as follows. At the beginning, we present the proposal of the system architecture. In the next section, the model domain of the ontology used at the Behavior and Performance Knowledge Bases are presented. Finally, an example of the use of the Behavior and Performance K.B. applied to web caching tier is explained.

2. Web Performance Engineering Limitations

Current web applications are quite different from first web systems, e.g. application complexity is higher than before. Nowadays, with the show up of web 2.0 applications, the behavior of users and system data flows have completely changed. Traditional web performance engineering techniques are not enough efficient since they do not consider the dynamic nature of web pages. Another drawback for web performance engineering is the absence of global information about the system in order to dynamically autoadjust its parameters.

The main objective of web performance engineering is to achieve a global quality of service (QoS). However, systems are usually monitored and analyzed only in a partial way. Each system tier gets performance information of a particular level and may change the behavior or configuration using only that information. But a performance increase in one part of the system can negatively impact the overall performance of the system, in direct contradiction to the knowledge of a particular tier. Thus, We need to achieve a global goal, but we only have tools to perform actions in partial system elements.

Besides, we have to face of the heterogeneity of web systems. Traditionally, web systems have been built in three-tier models even though more tiers could be defined and consequently increasing the heterogeneity. Moreover, managing information from the different parts of the system is also more complex.

Ontologies can help us to solve these drawbacks. By using ontologies, we propose to homogenize the information monitored from web system tiers and analyze it in a common way to apply local solutions to aim the global performance objective. Ontologies allow us to interchange knowledge between the different elements since they are formal and have semantics.

We propose the use of a knowledge base to store the global information of the system behavior and its performance. The knowledge base is implemented using the standard de facto ontology language. The main advantage of storing the global information about the system using a common expression is the simple way to analyze it. The formality of semantics should provide the necessary automation for extracting a global performance view. Ontologies

also provide an easier subtract for future reasoning. We may adventure the application of rules to produce more knowledge from the stored information.

Once the information of the system is analyzed, the setup of the different parts of the systems may change. The easy way to implement this changes is also to use a knowledge base where the configuration of the parts of the system is defined. This knowledge base could be implemented using some extension to existing modeling language for web as WSMML. There are a lot of work to be done about the modeling of web services and web applications with the use of ontologies, but the lack of work in the web architecture modeling is also important and this is the motivation of this proposal.

3. Proposed Architecture

In this paper we propose a centralized system to store and analyze the information about the behavior and performance of the different elements of web systems.

In Figure 1 we present a first approximation to the proposed architecture:

- *Behavior Knowledge Base*: This is the centralized module that uses an ontology to store information about all the significant elements in the web system, i.e., users, requests, proxies, caches nodes, gateways, web servers, database systems, etc. In Section 4 we present a complete explanation of the ontology model domain.
- *Web system elements*: Each module, tier, element or system in the web scenario has to share information with the Behavior Knowledge Base. Therefore, it is necessary to add some modules to the different web elements which analyze the process and measure their performance. Once this information is gathered, each module send it to the Behavior K.B. In Section 5.1 we present an example of the added module to a cache tier.

In this first approximation, only monitoring elements appear in our proposal. The main objective to reach in the future should have a complete monitoring and actuating architecture. Thus, We could add reasoners to the architecture that use the information at the Behavior K.B. to change the organization or even the way the system works.

We propose that the behavior of each module or element in the system will be modeled by an ontology in a knowledge base. Each of these ontologies has the information how the system works. Examples of information that system may use would be load balancing information, caches policies, task priorities, etc. Therefore, the reasoners use the information stored in the Behavior K.B. and changes the different Knowledge Bases of elements by application

of rules and operations (Figure 1). In the sections below a complete explanation about each module is presented.

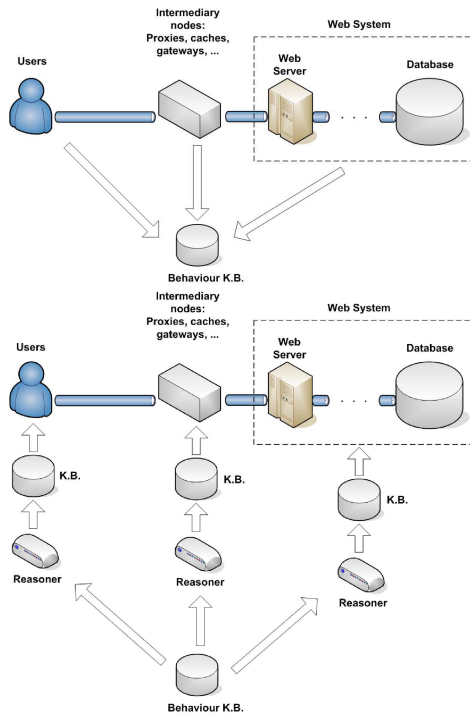


Figure 1. Behavior Knowledge Base Architecture and Reasoners and Actuators Architecture

3.1. Web System Elements Knowledge Base

In our proposed system changes over its configuration are produced. Therefore, the way the system works and its auto-setup has to be modeled. Thus, each part or element of the system will have associated its own information in the knowledge base.

We also have to use an ontological description for each part of the system due to their own characteristics and specific model domain. Semantic web and ontological languages are very active research fields, and there are a huge number of ontologies available ready to be used in these environments.

Once the different elements in the system are modeled through their corresponding ontologies, these elements will use the models to express the way they work.

3.2. Reasoner

The reasoner uses the information about the behavior of the clients, server tiers and server resources to determine the

best tuning of the web system. It uses heuristics, applying rules and operations, to setup the different elements in the web system.

When a state change of the system occurs (Transient State), the reasoner evaluates the new state. It analyzes the behavior and the performance of the different elements in this new state. After some period of time, if the performance has been improved, the reasoner considers that then new state as the stable (stationary) state. On the contrary, if the performance is worse, it recovers the last stable state and tries different changes in the setup of the system. The period of time between different evaluations of possible stable states has dynamic duration. Each time a stable state is restored after an evaluation period, the period of time for re-evaluation is consequently increased. Figure 2 shows a state diagram.

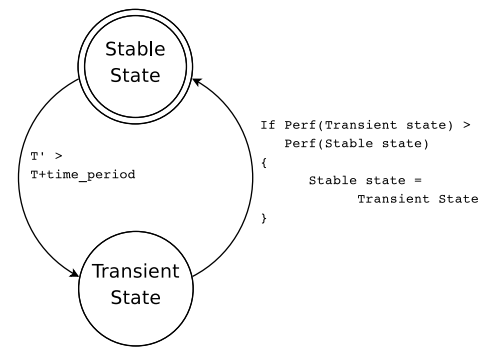


Figure 2. Reasoner operation state diagram

To create a new state (transient state) the reasoner applies operations. The reasoner operations are chosen analyzing the parameters in the Behavior K.B. The operations are different for each element of the web system. For example in [9] we defined the valid operations for a cache system, where the documents are fragmented to improve the performance. These operations are not valid for other elements of the web system.

3.3. System overhead

The proposed system needs to control, to analyze and to manage the information and behavior of the system. Thus, it is necessary new hardware and process elements independent from the web system hardware to avoid the overhead degradation of the quality of service of the web system. Therefore, the process associated to the knowledge bases and reasoners would be implemented by new hardware and computing tiers different to the functional tiers of the system.

Another drawback of our proposal is how to gather the information from the functional web system. We should need monitors adding the minimal overhead to listen events

and gather information from the processes, states or flows in the system. The overhead produced by these software monitors should be considered as negligible.

Once the data is gather from the different elements and web tiers, it should be sent to the centralized system where the knowledge bases and the reasoning is performed accordingly. The transmission overhead should be also small in comparison with the global load of each tier or element. In order to reduce the transmission overhead, the information could be sent as a summary in periodical time slots.

A future work is to study the minimum overhead over the functional elements ant tiers of the system needed to ensure the availability of our proposal.

4. Web Performance and Behavior Ontology

Ontologies, which are explicit formal descriptions of concepts in a domain, are composed of classes, class properties and restrictions on properties [16]. We use the ontology building life-cycle explained in [6, 17] and also used in other research works as [12, 13].

Our main research topic, as we have presented in Section 3, is the performance and behavior of the web elements at the application level. The elements which take importance at web scenario are: user sessions, user requests, HTTP requests, HTTP responses, etc.

4.1. Workload and HTTP Model Domain

Web applications and web systems are built over the HTTP protocol (Internet based application protocol). Therefore, the definition of our Ontology, which is used to represent the performance information, has to be determined by the definition of HTTP [7].

In a Web environment the servers are passive elements, and the users are the active ones. Users make requests to the servers, when they need services or information. These requests are translated to HTTP requests that are sent to the servers identified by an URI.

For users, the architecture of the server tiers are completely clear and that request is made to an URI [5]. Users don't worry about if that URI corresponds to an isolated server, a proxy server, a cache server, a load balancing server, etc. When the HTTP request arrives to the server identified in the URI, the corresponding server processes that request. The server process is different for each request. However, we classify the processes into different types: local tasks and remote tasks. The web-system could be structured into different tiers or levels. A HTTP request could be translated to one or more local tasks and in one or more remote tasks. Usually, the communication between the different tiers also utilizes HTTP as application layer protocol.

The local tasks, that a server uses to respond to user requests, generate a local workload on this tier of the web-system. We may identify different kinds of local workload for the different elements or components of the server: disks, processors, DB systems, memory, scripting interpreters modules, web server modules, etc. Some of these elements or components could need tasks associated to other tiers in the web-system. In these cases, new HTTP requests are generated between the different system tiers. These new requests can be translated to local workload in the target tier (remote tasks) and to network workload. Transmission times and capacity, latency and node process are components corresponding to the network workload [3]. In Figure 3 we present the model domain of concepts corresponding to workload. This model is used to create the ontology.

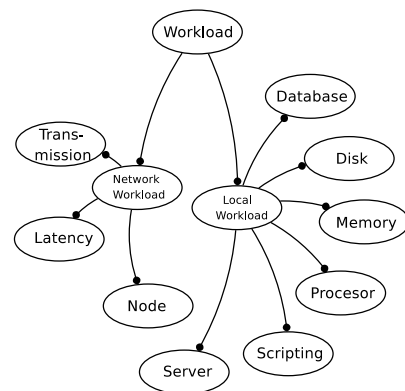


Figure 3. The abbreviate Workload model domain.

When local task in a tier is performed, a HTTP response is generated. If this HTTP response arrives to another tier, once all the responses will be arrived and all the local tasks will be done, another HTTP response is generated. This path is repeated right to the tier that received the user request (URI). The HTTP response generated by that last tier goes directly to the user. HTTP responses generate network workload, in the same way HTTP requests do. Figure 4 shows a simple model domain which corresponds to the HTTP protocol concepts.

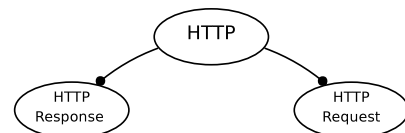


Figure 4. The abbreviate HTTP model domain.

4.2. Performance Model Domain

Our analysis is focused on the performance and behavior of the elements in a web system. Thus, we need to add classes to represent performance evaluation to the model domain. The main metrics for the different elements in a web system are also different. So we need to consider a wide range of metrics. In the model domain, each of those metrics corresponds to a subclass of the class representing its performance evaluation. In Figure 5 we see these subclasses. For each of these classes a different measurement system is used. It is not necessary to build a model for each of these measurement systems. We have taken advantage of one of the most useful features of OWL, knowledge sharing, and we have imported other important ontologies and their well-defined semantic knowledge to our model domain.

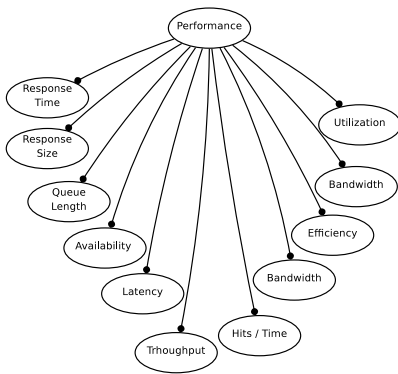


Figure 5. The abbreviate Performance Evaluation model domain.

We need to relate the performance evaluation classes with the workload classes. We should be able to represent relationships between tasks and their workload. In ontological languages, the properties are the elements that represent relationships between individuals. Figure 6 shows us object properties between the classes of our model domain, particularly for workload and performance classes.

4.3. Web Session Model Domain

Web users usually interact with the system, generating a continuous flow of requests as a result of the response information from the system. This set of requests is considered as a user session (Figure 7).

Figure 6 shows the object property between Web Session and HTTP classes. Each time a user generates a request, a HTTP request is associated to the server, service or data demands. The server answers through HTTP responses. As we explained at the beginning of Section 4.1, each HTTP message (response or request) generates workload over the

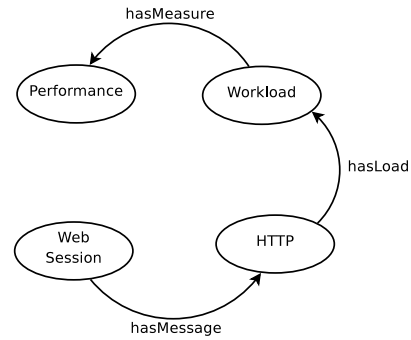


Figure 6. Object properties between classes.

network and over the different tiers in the web system. The object property between HTTP and Workload classes represents that fact in our model domain.

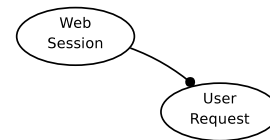


Figure 7. The abbreviate Web Session model domain.

To sum up, with the presented model domain, we represent the behavior and the performance of the different elements in a web system. This model could be used in some applications to improve these aspects. In the next section we present an scenario where the model has been used with to improve the web caching techniques.

5. Concept probe

This scenario needs information about the behavior and the performance of the different elements in the system. With this information, the system would cache the web pages in different ways, trying to get a better performance in the whole system. The ontology presented in the Section 4 has been used to model the behavior and the performance.

Typically web-based applications are based in tiered architecture. Usually in these applications the user interface, functional process logic (business rules), data storage and data access are developed as independent as possible, most often on separate platforms.

As the number of tiers and interfaces between tiers increase, the generation process is also longer and computationally more expensive. To minimized the workload generated by each user request response, a cache tier is been placed between clients and server tiers. Web caching also contributes to reduce bandwidth usage, server load, and user

perceived lag. The caching tier stores generated content to avoid the load over the server tiers at next request of the same page.

As is presented in [8], caching may be improved by reducing the minimum caching unit. Instead of caching whole pages, if pages fragments are cached independently, the performance of dynamic webs may be improved. But the way that fragments are built (fragment design) is essential to achieve better performance than in whole pages caching scenario. Moreover, the best fragment design, from the performance point of view, could change along time.

We propose an architecture which improves the caching techniques using fragment caching process. Thus, the Web Cache Tier needs to gather information about the behavior of the web system and the users. It is important to know the user profiles: pages most requested, navigation paths, thinking times, etc. From the server side is important to know the response time and the size of each request, the changing ratio of the web pages contents and, finally, the sharing ratio between web pages contents. These parameters are the most interesting to achieve the fragment design with the best performance.

We have implemented a Web Cache Tier which gathers information about the system and stores it in an Knowledge Base which uses the ontology defined at Section 4. As future work we will develop the reasoner and actuators that use that gathered information to change the way fragments are cached.

5.1. System Description

The usual architecture of the tiered web system consists on: (a) server side; (b) cache tier and (c) client side. The server side consists on a three tiers application (Figure 8). We experimented with WordPress [2], thus web logs are wide extended over Internet with higher update ratios.

WordPress is structured in three tiers: (a) MySQL as the database system; (b) PHP is the scripting language and (c) use of templates for the presentation and design of the HTML document.

Our main objective is to study the cache system. We need to introduce some monitors to gather information in this tier of the system. Since the most extended ontological tools are open programmed with Java and, for practical reasons, to avoid the implementation of the whole tier, we have chosen Smart Cache [11] for our experiments. Smart Cache is HTTP 1.1 compliant and HTTPS proxy cache written in pure Java 1.1.

The cache tier (Smart Cache tool) has been modified to gather information about the requests that serves to the clients, requests to the origin servers, measures the performance, etc. All that gathered information is use to create the model in the Ontological Language presented at Section 4.

We use Jastor [4] to integrate the Ontological Language and Java. Besides, we have used Jastor which is an automatic generator of Java interfaces, implementations, factories and listeners based on the properties and classes hierarchies in the Ontological Language.

Finally, we use Apache JMeter to emulate the users [1]. JMeter is a Java desktop application designed to test web functional behavior and to measure its performance. It may be used to test performance both on static and dynamic resources. It can be also used to simulate the server, network or object overload, to test their strength or to analyze their overall performance under different workload profiles and heavy concurrent load.

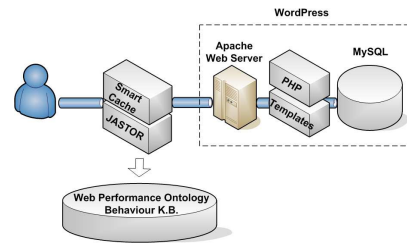


Figure 8. Concept Prove System Description.

5.2. Validation

We run the system with a synthetic workload (benchmark) to evaluate all the system. The synthetic workload corresponds to one hundred concurrently users requesting random pages to the server. After the runtime period of one million requests, we validate the behavior and performance information of the different elements in the system with the values observed and gathered by the cache system and the behavior knowledge base.

In this first experiment we only gather information about the individual user requests: number of requests, response times and response sizes. Both in the user side and in the origin server side, we logged the activity in the system. Apache provides modules to create web logs in a standardized text file format. Common Logfile Format (CLF) is the most usual log format. CLF, for each request received by the server, add a new line in the text file. With the information stored in that text line we can extract the number of requests and the size of that requests. Additionally, thanks to JMeter, we can extract a huge number of measures and statistics, in particular, response sizes, response times and requests number.

Once we have analyzed the Apache and the JMeter logs, we compare these measurements with the ones gathered by the cache level in the knowledge base. In that way, we realized that the values are exactly the same, with the ex-

ception of response times, because there is a small network overhead time between the cache system and the Jmeter. So that we can conclude the Ontology defined in Section 4 may be used to model the behavior and performance of web systems.

6. Conclusion

An architecture to improve web performance has been proposed. This architecture is based on web performance improvement by changing the configuration and setup of the system during runtime period. These changes are made at runtime period because the parameters that condition the performance and behavior of the system are not known in the development phase and they should change along time.

An ontological system is proposed to analyze the behavior of the system and to make fragments of information trying to improve performance. The model domain of an ontology to reach that aim has been presented. A knowledge base to store information about the behavior and performance of the system has been create using that ontology.

That K.B. has been tested in a case of study presented in the last section of the paper. The case of study is based on testing and monitoring the information in the web cache tier. Smartcache has been used as cache system, and has been modified to gather performance information. Jastor has been used to integrate the ontological language in the cache system. Therefore, the cache tier has been available to gather information from the system and store those information in the K.B.

In a validation phase the information stored in the K.B. and the information store in the traditional logs of the different tools of the web system have been compared. Results are promising since they validated the process of gathering information and the use of the ontological language.

Future work is open to two different branches. First, to create the specific ontological language for modeling the different elements in the web system and the integration of that ontologies in the different elements that would use the store information in the corresponding K.B.

Second, the definition of the operations and rules that change the models of each element in the web system by using the information in the performance and behavior knowledge base.

Acknowledgments

This work is partially financed by the Spanish Ministry of Education and Science through TIN2007-29683-E project.

References

- [1] Apache. Apache jmeter. <http://jakarta.apache.org/jmeter/>, 2007.
- [2] Automatic. Wordpress 2.0. <http://wordpress.com/>, 2007.
- [3] P. Baldi, P. Frasconi, and P. Smyth. *Modeling the Internet and the Web: Probabilistic Methods and Algorithms*. Wiley, 2003.
- [4] J. B. Ben Szekely, Rob Gonzalez. Jastor. <http://jastor.sourceforge.net/>, 2007.
- [5] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax. RFC 3986, Internet Engineering Task Force, January 2005.
- [6] J. Davies, F. van Harmelen, and D. Fensel, editors. *Towards the Semantic Web: Ontology-driven Knowledge Management*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
- [7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, Internet Engineering Task Force, June 1999.
- [8] C. Guerrero, C. Juiz, and R. Puigjaner. The applicability of balanced esi for web caching. In *Proceedings of the 3rd International Conference on Web Information Systems and Technologies*, 2007.
- [9] C. Guerrero, C. Juiz, and R. Puigjaner. Dynamic web fragment architecture. *Actas de Talleres de ingenieria del software y bases de datos*, 1(3):3–11, 2007.
- [10] P. Killelea. *Web Performance Tuning*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.
- [11] R. Kolar. Smart cache 0.93. <http://scache.sourceforge.net/>, 2007.
- [12] I. Lera, C. Juiz, and R. Puigjaner. Performance-related ontologies and semantic web applications for on-line performance assessment intelligent systems. *Sci. Comput. Program.*, 61(1):27–37, 2006.
- [13] I. Lera, P. P. Sancho, C. Juiz, R. Puigjaner, J. Zottl, and G. Haring. Performance assessment of intelligent distributed systems through software performance ontology engineering (spoe). *Software Quality Control*, 15(1):53–67, 2007.
- [14] D. A. Menasce and V. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
- [15] D. A. Menasce and A. F. A. Virgilio. *Scaling for E Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
- [16] M. K. Smith, C. Welty, and D. L. McGuinness. Owl web ontology language guide. W3c recommendation, World Wide Web Consortium, February 2004.
- [17] M. Uschold. Towards a methodology for building ontologies, 1995.