An Application Server for the Semantic Web

Daniel Oberle, Steffen Staab, Raphael Volz
University of Karlsruhe
Institute for Applied Informatics and Formal Description Methods
76128 Karlsruhe
lastname@aifb.uni-karlsruhe.de

ABSTRACT

The Semantic Web relies on the complex interaction of several technologies involving ontologies. Therefore, sophisticated Semantic Web applications typically comprise more than one software module. Instead of coming up with proprietary solutions, developers should be able to rely on a generic infrastructure for application development in this context. We call such an infrastructure Application Server for the Semantic Web whose design and development are based on existing Application Servers. However, we apply and augment their underlying concepts for use in the Semantic Web and integrate semantic technology within the server itself. We provide a short overview of requirements and design issues of such a server and present our implementation and ongoing work KAON SERVER.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Extensibility*; D.2.11 [**Software Engineering**]: Software Architectures; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based Services*

Keywords

Application Server, Extensibility, Interoperation, KAON SERVER, Ontology, Reuse, Semantic Middleware, Semantic Web

1. INTRODUCTION

Ontologies serve various needs in the Semantic Web, like storage or exchange of data corresponding to an ontology, ontology-based reasoning or ontology-based navigation. Building a complex Semantic Web application, one may not rely on a single software module to deliver all these different services. The developer of such a system would rather want to easily combine different — preferably existing — software modules.

So far, however, such integration of ontology-based modules had to be done ad-hoc, generating a one-off endeavour, with little possibilities for re-use and future extensibility of individual modules or the overall system.

We present an infrastructure that facilitates plug'n'play engineering of ontology-based modules and, thus, the development and maintenance of comprehensive Semantic Web applications, an infrastructure which we call *Application Server for the Semantic Web (ASSW)*. Existing Application Servers typically comprise functionality like connectivity and security, flexible handling of software modules, monitoring, transaction processing etc. They are the middleware between browser-based front-ends and back-end databases and legacy systems. The Application Server for the Semantic Web will help to put the Semantic Web into practice because it adopts and augments this idea for easier development of Semantic Web applications. In addition, semantic technology is used within the server itself what allows us to achieve an even greater functionality than existing Application Servers.

The following sections talk about requirements and design decisions leading to the conceptual architecture of an Application Server for the Semantic Web (for a detailled discussion cf. [4]). Finally, we describe our implementation effort, called KAON SERVER, which is currently work in progess.

WWW2004, May 17–22, 2004, New York, NY USA.
ACM xxx.xxx.

2. REQUIREMENTS

The requirements on an Application Server for the Semantic Web can be divided in four groups. First, such a server should meet requirements that are common to existing Application Servers (connectivity, ease of use, offering functionality via different communication protocols, security). Second, another group of requirements comprises flexible handling of modules (extensibility, integrating existing functionality via different communication protocols, expressing and managing dependencies). Third there are requirements that are specific to the Semantic Web like language support, semantic interoperation, ontology mapping and modularisation, finding, accessing, modifying and storing of ontologies, transactions and rollbacks, evolution and versioning, monitoring, inferencing and verification. The last group comprises requirements for semantic enhancement of the Application Server allowing discovery of software modules and APIs, classification of software modules and facilitating implementation tasks.

While the common requirements are met by most of the existing Application Servers, Semantic Web specific requirements and the ones that call for the semantic enhancement of the server itself are clearly beyond state-of-the-art.

3. DESIGN

In order to meet the requirement "flexible handling of modules", the Microkernel design pattern is our basic design choice. The pattern applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and application-specific parts. In our setting, the Microkernel's minimal functionality must take the form of simple management operations, i.e. starting, initializing, monitoring, combining and stopping of software modules as well as dispatching of messages between them.

This approach requires software modules to be uniform so that they can be treated equally by the Microkernel. Hence, existing

software modules have to be made deployable, i.e. they have to be wrapped for plugging them into the Microkernel. Thus, in our terminology, a software module becomes a *component*.

All components are equal as seen from the Microkernel's perspective. Hence, in order to allow a client discovering the components it is in need of, we have to distinguish between them. A registry is necessary to store descriptions of all deployed components. We came up with a management ontology [3] that responds to the requirements for semantic enhancement of the server. It conceptually distinguishes components into functional (ones that are of interest to the client, e.g. an RDF store), proxy (a special type of functional component) and system components (ones that provide functionality for the server itself, e.g. the registry).

The resulting design elements of the architecture are divided into Connectors, Management Core, Interceptors and Functional Components, like depicted in Figure 1.

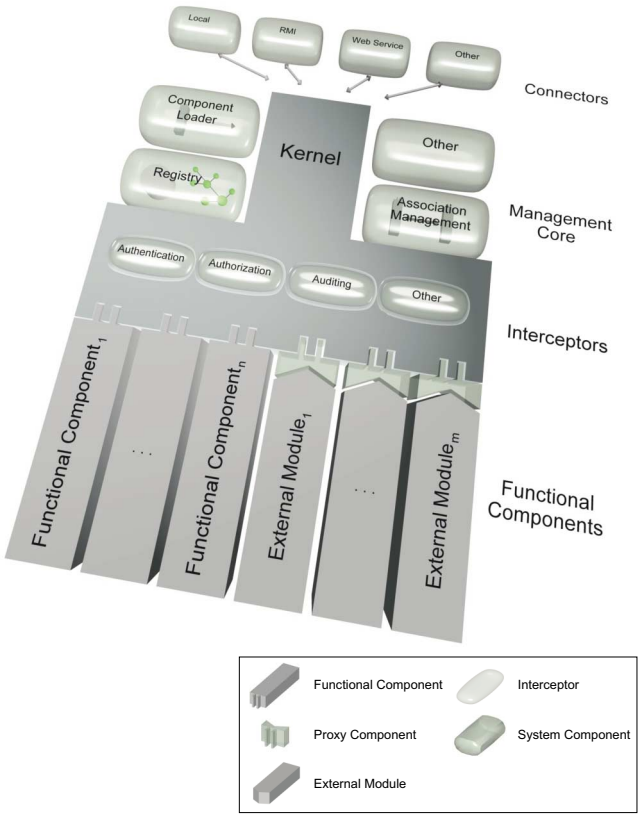

Figure 1: Conceptual Architecture

Surrogates

Surrogates (not shown in Figure 1) are objects embedded in the client application that relieve the developer of the communication details similar to stubs in CORBA (cf. requirement "Ease of use"). They offer the same API as a particular component and relay communication to any connector which in turn passes the request to the respective functional component through the Microkernel.

Connectors

Connectors are system components. They send and receive requests and responses over the network. Connectors could also allow to publish components' methods as separate web services with automatically generated DAML-S descriptions out of the registry. Offering the functionality with peer or agent protocols is also possible (cf. requirement "Offering functionality via different communication protocols").

Management Core

The Management Core comprises the Microkernel as well as several system components. It is required to deal with the discovery, allocation and loading of components. The registry, an ontology store, manages descriptions of the components and facilitates the discovery of a functional component for a client. The component loader facilitates the deployment process for a client. It takes a component description as argument, handles the deployment, enters the description in the registry and applies the association management if necessary. The latter is another system component that puts ontological associations between components into action. E.g., event listeners can be put in charge so that a component A is notified when B issues an event.

Interceptors

Interceptors are software entities that monitor a request and modify it before the request is sent to the component. A component can be deployed with a stack of arbitrary interceptors. Security aspects or semantic interoperation are met by interceptors.

Functional Components

RDF stores, ontology stores etc., are finally deployed to the management kernel as functional components. In combination with the component loader, the registry can start functional components dynamically on client requests. Proxy components can be developed for external modules that cannot be made deployable. Most Semantic Web specific requirements are met by functional components.

4. IMPLEMENTATION

We are currently implementing the aforementioned architecture in a system called KAON SERVER which is part of the KArlsruhe Ontology and Semantic Web Toolsuite (KAON) [1]. The KAON SERVER is developed in the context of WonderWeb [2], an EU IST funded project, whose aims are, among others, a tight integration of existing tools like ontology editors, stores and inference engines.

In the case of the KAON SERVER, we use the Java Management Extensions (JMX) — an open technology and currently the state-of-the-art for component management. Basically, JMX defines interfaces of managed beans, or *MBeans* for short, which are JavaBeans that represent JMX manageable resources. MBeans are hosted by an *MBeanServer* which allows their manipulation. All management operations performed on the MBeans are done through interfaces on the MBeanServer. In our setting, the MBeanServer realizes the kernel and MBeans realize components.

5. REFERENCES

[1] KAON - the KArlsruhe Ontology and Semantic Web Toolsuite. `http://kaon.semanticweb.org`.

[2] WonderWeb - Ontology Infrastructure for the Semantic Web. `http://wonderweb.semanticweb.org`.

[3] D. Oberle, M. Sabou, D. Richards, and R. Volz. An ontology for semantic middleware: extending daml-s beyond web-services. In *OTM 2003 Workshops*, volume 2889 of *LNCS*, 2003.

[4] D. Oberle, S. Staab, R. Studer, and R. Volz. Supporting application development in the semantic web. *ACM Transactions on Internet Technology (TOIT)*, 2005. to appear.