An extensible ontology software environment

Daniel Oberle[1], Raphael Volz[1,2], Steffen Staab[1], and Boris Motik[2]

[1] University of Karlsruhe, Institute AIFB
D-76128 Karlsruhe, Germany
email: {lastname}@aifb.uni-karlsruhe.de
[2] FZI - Research Center for Information Technologies
D-76131 Karlsruhe, Germany
email: {lastname}@fzi.de

Summary. The growing use of ontologies in applications creates the need for an infrastructure that allows developers to more easily combine different software modules like ontology stores, editors, or inference engines towards comprehensive ontology-based solutions. We call such an infrastructure Ontology Software Environment. The article discusses requirements and design issues of such an Ontology Software Environment. In particular, we present this discussion in light of the ontology and (meta)data standards that exist in the Semantic Web and present our corresponding implementation, the KAON SERVER.

1 Introduction

Ontologies are increasingly being applied in complex applications, e.g. for Knowledge Management, E-Commerce, eLearning, or information integration. In such systems ontologies serve various needs, like storage or exchange of data corresponding to an ontology, ontology-based reasoning or ontology-based navigation. Building a complex ontology-based system, 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.

This paper is about an infrastructure that facilitates plug'n'play engineering of ontology-based modules and, thus, the development and maintenance of comprehensive ontology-based systems, an infrastructure which we call an *Ontology Software Environment*. The Ontology Software Environment facilitates re-use of existing ontology stores, editors, and inference engines. It combines means to coordinate the information flow between such modules, to define

dependencies, to broadcast events between different modules and to translate between ontology-based data formats.

Communication between modules requires ontology languages and formats. The Ontology Software Environment presented in this paper supports languages defined in the Semantic Web because they are currently becoming standards specified by the World Wide Web Consortium (W3C) and thus will be of importance in the future. We introduce the term of an *Application Server for the Semantic Web (ASSW)*, which is a particular type of Ontology Software Environment, especially designed for supporting the development of Semantic Web applications.

The article is structured as follows: First, we provide a brief overview about the Semantic Web and its languages in section 2 and motivate the need for an Application Server for the Semantic Web by a scenario in section 3. We derive requirements for such a server in section 4. Sections 5 and 6 describe the design decisions that immediately answer to important requirements, namely extensibility and discovery. The conceptual architecture is then provided in section 7. Section 8 presents the KAON SERVER, a particular Application Server for the Semantic Web which has been implemented. Related work and conclusions are given in sections 9 and 10, respectively.

2 The Semantic Web

In this section we want to introduce the reader to the architecture and languages of the Semantic Web that we use in our Application Server. The Semantic Web augments the current WWW by adding machine understandable content to web resources. Such added contents are called metadata whose semantics can be specified by making use of ontologies. Ontologies play a key-role in the Semantic Web as they provide consensual and formal conceptualizations of domains, enabling knowledge sharing and reuse.

The left hand side of figure 1 shows the static part of the Semantic Web[3], i.e. its language layers. Unicode, the URI and namespaces (NS) syntax and XML are used as a basis. XML's role is limited to that of a syntax carrier for data exchange. XML Schema defines simple data types like string, date or integer.

The Resource Description Framework (RDF) may be used to make simple assertions about web resources or any other entity that can be named. A simple assertion is a statement that an entity has a property with a particular value, for example, that this article has a title property with value "An extensible ontology software environment". RDF Schema extends RDF by class and property hierarchies that enable the creation of simple ontologies.

[3] Semantic Web - XML 2000, Tim Berners-Lee,
http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html

The Ontology layer features OWL (Web Ontology Language[4]) which is a family of richer ontology languages that augment RDF Schema. OWL Lite is the simplest of these. It is a limited version of OWL Full that enables simple and efficient implementation. OWL DL is a richer subset of OWL Full for which reasoning is known to be decidable so complete reasoners may be constructed, though they will be less efficient than an OWL Lite reasoner. OWL Full is the full ontology language which is undecidable.

The Logic layer will provide an interoperable language for describing the sets of deductions one can make from a collection of data – how, given a ontology-based information base, one can derive new information from existing data[5].

The Proof language will provide a way of describing the steps taken to reach a conclusion from the facts. These proofs can then be passed around and verified, providing short cuts to new facts in the system without having each node conduct the deductions themselves.

The Semantic Web's vision is that once all these layers are in place, we will have a system in which we can place trust that the data we are seeing, the deductions we are making, and the claims we are receiving have some value. The goal is to make a user's life easier by the aggregation and creation of new, trusted information over the Web[6]. The standardization process has currently reached the Ontology layer, i.e. Logic, Proof and Trust layers aren't specified yet.

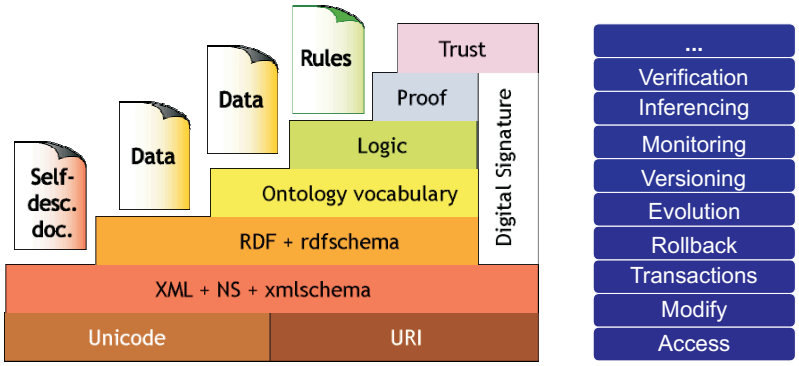

Fig. 1. Static and dynamic aspects of the Semantic Web layer cake

[4] W3C Working Draft, http://www.w3.org/TR/owl-ref

[5] A better description of this layer would be "Rule layer", as the Ontology layer already features a logic calculus with reasoning capabilities. However, we want to conform to the official naming here.

[6] Building the Semantic Web, Edd Dumbill, http://www.xml.com/pub/a/2001/03/07/buildingsw.html

The right hand side of figure 1 depicts the Semantic Web's dynamic aspects that apply to data across all layers. Often, the dynamic aspects are neglected by the Semantic Web community, however, from our point of view, they are an inevitable part for putting the Semantic Web into practice. It is obvious that there have to be means for access and modification of Semantic Web data. According to the the well-known ACID (atomicity, consistency, independence, durability) of Database Management System (DBMS), transactions and rollbacks of Semantic Web data operations should also be possible. Evolution and versioning of ontologies are an important aspect, because ontologies usually are subject to change [24]. Like in all distributed environments, monitoring of data operations becomes necessary for security reasons. Finally, reasoning engines are to be applied for the deduction of additional facts[7] as well as for semantic validation.

3 A Motivating Scenario

This section motivates the needs for the cooperation and integration of different software modules by a scenario depicted in figure 2. The reader may note that some real-world problems have been abstracted away for the sake of simplicity.

Assume a simple genealogy application. Apparently, the domain description, viz. the ontology, will include concepts like Person and make a distinction between Male and Female. There are several relations between Persons, e.g. hasParent or hasSister. The domain description can be easily expressed with OWL DL, e.g. Person subsumes both Male and Female concepts. However, many important facts that could be inferred automatically have to be added explicitly. E.g., information about the parents' brothers of a person are sufficient to deduce her or his uncles. A rule-based system is needed to capture such facts automatically. Persons will have properties that require structured data types, such as dates of birth, which should be syntactically validated. Such an ontology could serve as the conceptual backbone and information base of a genealogy portal. It would simplify the data maintenance and offer machine understandability. To implement the system, all the required modules, i.e. a rule-based inference engine, a DL reasoner, a XML Schema data type verifier, would have to be combined by the client applications themselves. While this is a doable effort, possibilities for re-use and future extensibility hardly exist.

The application demands from an Application Server for the Semantic Web is to hook up to all the software modules and to offer management of data flow between them. This also involves propagation of updates and rollback behavior, if any module in the information chain breaks. In the following

[7] E.g., if "cooperatesWith" is defined as a symmetric property in OWL DL between persons A and B, a reasoner can deduce that B cooperatesWith A, too.

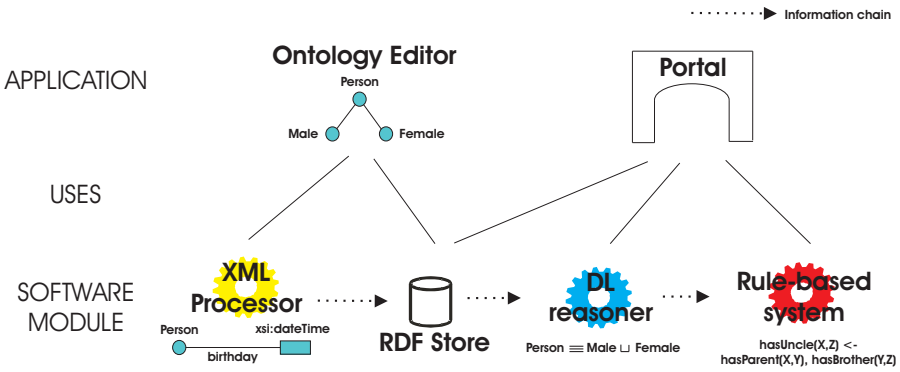

Fig. 2. Software modules in the genealogy applications

section, we will discuss requirements for an Application Server for the Semantic Web in more detail. The requirements are derived from the scenario as well as from the static and dynamic parts of the Semantic Web.

4 Requirements

Basically, the we can establish four groups of requirements. First, an Application Server for the Semantic Web should respond to the static aspects of the Semantic Web layer cake. In particular, it has to be aware of all Semantic Web languages. The need to translate between the different languages also belongs to the static aspects. Such a translation increases interoperability between existing software modules that mostly focus on one language only. Second, the dynamic aspects result in another group of requirements, viz. finding, accessing and storing of data, consistency, concurrency, durability and reasoning. Third, clients may want to connect remotely to the Application Server for the Semantic Web and must be properly authorized. Hence, a distributed system like the Semantic Web needs connectivity and security. Finally, the system is expected to facilitate a extensible and reconfigurable infrastructure. The last group of requirements therefore deals with flexible handling of modules. In the following paragraphs we will investigate on the requirements.

- **Requirements stemming from the Semantic Web's static part**
  - *Language support* One requirement is the support of all the Semantic Web's ontology and metadata standards. The Application Server for the Semantic Web has to be aware of RDF, RDFS, OWL as well as future languages that will be used to specify the logic, proof and trust layers.
  - *Semantic Interoperation* We use the term Semantic Interoperation in the sense of translating between different ontology languages with different semantics. At the moment, several ontology languages populate

the Semantic Web. Besides proprietary ones, we have already mentioned RDFS, OWL Lite, OWL DL and OWL Full before. Usually, ontology editors and stores focus on one particular language and are not able to work with others. Hence, an Application Server for the Semantic Web should allow to translate between different languages and semantics.
  - *Ontology Mapping* In contrast to Semantic Interoperation, Ontology Mapping translates between different ontologies of the same language. Mapping may become necessary as web communities usually have their own ontology and could use Ontology Mapping to facilitate data exchange.
- **Requirements stemming from the Semantic Web's dynamic part**
  - *Finding, accessing and storing of ontologies* Semantic Web applications like editors or portals have to access and finally store ontological data. In addition, the development of domain ontologies often requires integration of other ontologies as starting point. Examples are Wordnet or top-level ontologies for the Semantic Web [4]. Those could be stored and offered by an Application Server for the Semantic Web to editors.
  - *Consistency* Consistency of information is a requirement in any application. Each update of a consistent ontology must result in an ontology that is also consistent. In order to achieve that goal, precise rules must be defined for ontology evolution [24]. Modules updating ontologies must implement and adhere to these rules. Also, all updates to the ontology must be done within transactions assuring the properties of atomicity, consistency, isolation and durability (ACID).
  - *Concurrency* It must be possible to concurrently access and modify Semantic Web data. This may be achieved using transactional processing, where objects can be modified at most by one transaction at the time.
  - *Durability* Like consistency, durability is a requirement that holds in any data-intense application area. It may be accomplished by reusing existing database technology.
  - *Reasoning* Reasoning engines are core components of semantics-based applications and can be used for several tasks like semantic validation and deduction. An Application Server for the Semantic Web should provide access to such engines, which can deliver the reasoning services required.
- **Connectivity and Security**
  - *Connectivity* An Application Server for the Semantic Web should enable loose coupling, allowing access through standard web protocols, as well as close coupling by embedding it into other applications. A client may want to use the system locally or connect to it remotely via web services, for instance.
  - *Security* Guaranteeing information security means protecting information against unauthorized disclosure, transfer, modification, or destruc-

tion, whether accidental or intentional. To realize it, any operation should only be accessible by properly authorized clients. Proper identity must be reliably established by employing authentication techniques. Confidential data must be encrypted for network communication and persistent storage. Finally, means for monitoring (logging) of confidential operations should be present.

- **Flexible handling of modules**
  - *Extensibility* The need for extensibility applies to most software systems. Principles of software engineering avoid system changes when additional functionality is needed in the future. Hence, extensibility is also desirable for an Application Server for the Semantic Web. In addition, such a server has to deal with the multitude of layers and data models in the Semantic Web that lead to a multitude of software modules, e.g. XML parsers or validators that support the XML Schema datatypes, RDF stores, tools that map relational databases to RDFS ontologies, ontology stores and OWL reasoners. Therefore, extensibility regarding new data APIs and corresponding software modules is an important requirement for such a system.
  - *Discovery of software modules* For a client, there should be the possibility to state precisely what it wants to work with, e.g. an RDF store that holds a certain RDF model and allows for transactions. Hence, means for intelligent discovery of software modules are required. Based on a semantic description of the search target, the system should be able to discover what a client is looking for.
  - *Dependencies* The system should allow to express dependencies between different software modules. For instance, that could be the setting up of event listeners between modules. Another example would be the management of a dependency like "module A is needed for module B".

In the following sections 5 to 7, we develop an architecture that is a result from the requirements put forward in this section. After that we present the implementation details of our Application Server for the Semantic Web called KAON SERVER.

5 Component Management

Due to the requirement for extensibility, we decided to use the Microkernel design pattern. 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. The Microkernel also serves as a socket for plugging in these extensions and coordinating their collaboration [14].

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 plus dispatching of messages between them. This approach requires software modules to be uniform so that they can be treated equally by the kernel. Hence, in order to use the Microkernel, software modules that shall be managed have to be brought into a certain form. We call this process *making existing software deployable*, i.e. bringing existing software into the particular infrastructure of the Application Server for the Semantic Web, that means wrapping it so that it can be plugged into the Microkernel. Thus, a software module becomes a *deployed component*. The word *deployment* stems service management and service oriented architectures where it is a terminus technicus. We adopt this meaning and apply it in our setting. We refine it as the process of registering, possibly initializing and starting a component to the Microkernel.

Apart from the cost of making existing software deployable, the only drawback of this approach is that performance will suffer slightly in comparison to stand alone use, as a request has to pass through the kernel first (and possibly the network). A client that wants to make use of a deployed component's functionality talks to the Microkernel, which in turn dispatches requests.

But besides the drawbacks mentioned above, the Microkernel and component approach delivers several benefits. By making existing functionality, like RDF stores, inference engines etc., deployable, one is able to treat everything the same. As a result, we are able to deploy and undeploy components ad hoc, reconfigure, monitor and possibly distribute them dynamically. Proxy components can be developed for software that cannot be made deployable for whatever reasons. Throughout the paper, we will show further advantages, among them

- enabling a client to discover the component it is in need of (cf. section 6)
- definition of dependencies between components (cf. section 7)
- easy realization of security, auditing, trust etc. as interceptors (further discussed in section 7)
- incorporation of quality criteria as attributes of a component in registry (cf. section 10)

Thus, we responded to the requirement of extensibility. In the following, we discuss how the discovery of software modules can be achieved.

6 Description of Components

This section responds to the requirement "discovery of software modules". As pointed out in the section 5, all components are equal as seen from the kernel's perspective. In order to allow a client to discover the components it is in need of, we have to make their differences explicit. Thus, there is a need of a registry that stores descriptions of all deployed components. In this

section we show how a description of a component may look like. We start with the definition of a component and then specialize. The definitions result in a taxonomy that is primarily used to facilitate component discovery for the application developer.

Component Software module which is deployed to the kernel.
System Component Component providing functionality for the Application Server for the Semantic Web, e.g. a connector.
Functional Component Component that is of interest to the client and can be looked up. Ontology-related software modules become functional components by making them deployable, e.g. RDF stores.
External Module An external module cannot be deployed directly as it may be programmed in a different language, live on a different computing platform, uses interfaces unknown, etc. It equals a functional component from a client perspective. This is achieved by having a proxy component deployed relaying communication to the external module.
Proxy Component Special type of component that manages the communication to an external module. Examples are proxy components for inference engines, like FaCT [15].

Each component can have *attributes* like the name of the interface it implements, connection parameters or other low-level properties. Besides, we want to be able to express *associations* between components. Associations can be dependencies between components, e.g. an ontology store component can rely on an RDF store for actual storage, or event listeners etc. Associations will later be put in action by the association management component (cf. section 7).

We formalize taxonomy, attributes and associations in a management ontology like outlined in figure 3 and table 1[8]. The ontology formally defines which attributes a certain component may have and places components into a taxonomy. In the end, actual functional components like KAON's RDF Server and the Engineering Server (cf. subsection 8.4) would be instantiations of RDFStore and OntologyStore, respectively.

Our approach allows us to realize the registry itself as a component. As explained in section 5, the Microkernel manages any functionality as long as it conforms to the contract. The registry is not of direct interest to the client - it is only used to facilitate the discovery of functional and proxy components. Hence, we can declare it as an instance of system component.

So far we have discussed two requirements, viz. extensibility and discovery of software components, which led to fundamental design decisions. The next section focuses on the conceptual architecture.

[8] The table shows some exemplary properties of the concept "Component". We use the term property as generalization for attribute and association. An attribute's range always is a string, whereas associations relate two concepts.

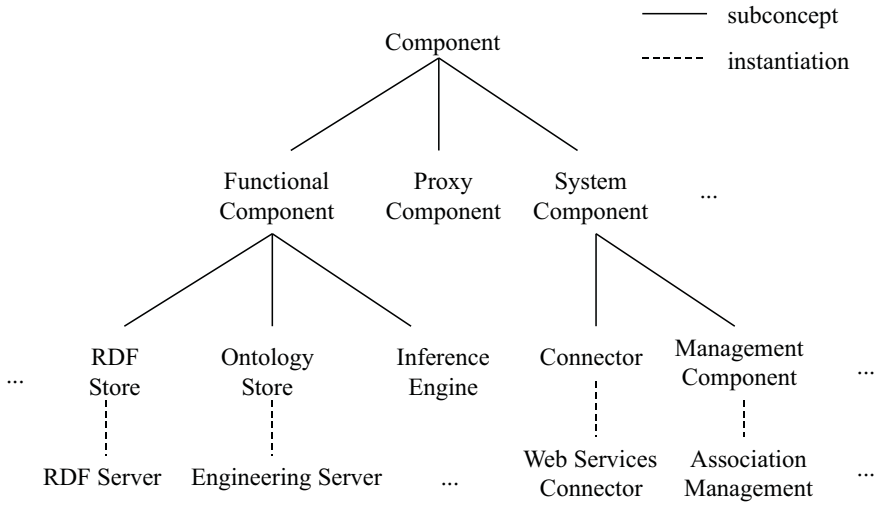

Fig. 3. Taxonomy of components

Concept	Property	Range
Component	Name	String
	Interface	String
	...	...
	receivingEventsFrom	Component
	sendingEventsTo	Component
	dependsOn	Component
	...	...

Table 1. Attributes and associations of Component

7 Conceptual Architecture

When a client connects to the Application Server for the Semantic Web it discovers the functional components it is in need of. That could be an RDF store or an inference engine etc. The system tries to find a deployed functional component in the registry fulfilling the stated requirements and returns a reference.

Surrogates for the functional component on the client side can handle the communication over the network. The counterpart to the surrogate on the server side is a connector component. It maps requests to the kernel's methods. All requests finally pass the management kernel which dispatches them to the actual functional component. While dispatching, the properness of a request can be checked by interceptors that may deal with authentication, authorization or auditing. An interceptor is a software entity that looks at a request and modifies it before the request is sent the component. Finally, the

response passes the kernel again and finds its way to the client through the connector.

After this brief procedural overview, the following paragraphs will explain the architecture depicted in figure 4. Note that in principle, there will be only three types of software entities: components, interceptors and the kernel. Components are specialized into functional, system and proxy components to facilitate the discovery for the application developer.

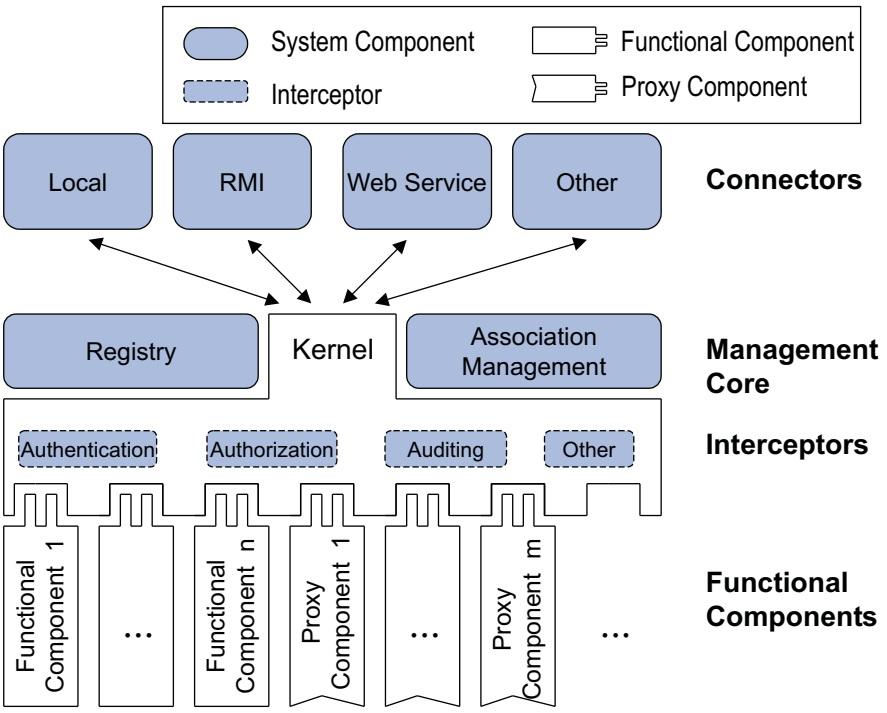

Fig. 4. Conceptual Architecture

Connectors

Connectors are system components. They send and receive requests and responses over the network by using some protocol. Apart from the option to connect locally, further connectors are possible for remote connection: e.g. ones that offer access per Java Remote Method Invocation (RMI), or ones that offer access per Web Service. Counterparts to a connector on the client side are surrogates for functional components that relieve the application developer of the communication details similar to stubs in CORBA.

Management Core

The Management Core comprises the Microkernel (also called management kernel or simply kernel in the following) as well as several system components. The Management Core is required to deal with the discovery, allocation and loading of components. The registry is a system component and hierarchically orders descriptions of the components. It thus facilitates the discovery of a functional component for a client (cf. section 6). Another system component called association management allows to express and manage relations between components. E.g., event listeners can be put in charge so that a component A is notified when B issues an event or a component may only be undeployed if others don't rely on it. The Management Core is extensible such that additional functionality may be provided by deploying new system components.

Interceptors

Interceptors are software entities that look at a request and modify it before the request is sent to the component. Security is realized by interceptors which guarantee that operations offered by functional components (including data update and query operations) in the server are only available to appropriately authenticated and authorized clients. A component can be registered with a stack of interceptors in the kernel. Sharing generic functionality such as security, logging, or concurrency control requires less work than developing individual component implementations.

Functional Components

RDF stores, ontology stores etc., are deployed to the management kernel as functional components (cf. section 5). In combination with the registry, the kernel can start functional components dynamically on client requests.

Table 2 shows where the requirements put forward in section 4 are reflected in the architecture. Note that most requirements are met by functional components. That is because the conceptual architecture presented here is generic, i.e. we could make almost any existing software deployable and use the system in any domain, not just in the Semantic Web. In the following section we discuss a particular implementation, KAON SERVER, that realizes functional components specific for Semantic Web standards.

Requirement \ Design Element	Connectors	Kernel	Registry	Interceptors	Association Management	Functional Components
Language Support						×
Semantic Interoperation						×
Ontology Mapping						×
Finding, accessing, storing of ontologies			×			×
Consistency						×
Concurrency		×				×
Durability						×
Reasoning						×
Connectivity	×					
Security				×		×
Extensibility		×				×
Discovery			×			
Dependencies					×	

Table 2. Reflections of the requirements in the architecture

8 Implementation

This section presents our implementation of an Application Server for the Semantic Web, called KAON SERVER. KAON SERVER offers a uniform infrastructure to host functional components, in particular those provided by the KAON project.

The KAON SERVER architecture reflects the conceptual architecture presented in the previous section. In the following, an in-depth description is given. We will start with the Management Core in 8.1 as it is necessary to understand Connectors in 8.2, Interceptors in 8.3 and Functional Components in 8.5. Several of the latter are implementations of the two Data APIs defined in the KAON Tool suite which are discussed before in subsection 8.4.

8.1 Management Core

The Management Core of an Application Server for the Semantic Web consists of the management kernel, the registry and association management system components. We will outline all of their implementations in the subsections below.

Kernel

In the case of the KAON SERVER, we use the Java Management Extensions (JMX[9]) as it is an open technology and currently the state-of-the-art for component management.

Java Management Extensions represent a universal, open technology for management and monitoring. By design, it is suitable for adapting legacy systems and implementing management solutions. Basically, JMX defines interfaces of managed beans, or *MBeans* for short, which are JavaBeans[10] 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 implements the kernel and MBeans implement the components. Speaking in terms of JMX, there is no difference between a system component and a functional component. Both are MBeans that are only distinguished by the registry.

Registry

We implemented the registry as MBean and re-used one of the KAON modules which have all been made deployable (cf. subsection 8.4). The main-memory implementation of the KAON API holds the management ontology. When a component is deployed, its description (usually stored in an XML file) is reflected by an instance of the proper concept. A client can use the KAON API's query methods to discover the component it is in need of.

Association Management

The management ontology allows to express associations between components. E.g., a dependency states that a given component requires the existence of another component. Therefore the server has to load all required components and be aware of the dependencies when unloading components. This essentially requires to maintain the number of clients to a component. A component can only be unloaded, if it does not have any further clients.

The JMX specification does not define any type of association management aspect for MBeans. That is the reason why we had to implement this functionality separately as another MBean. Apart from dependencies, it is able to register and manage event listeners between MBeans A and B, so that B is notified whenever A issues an event.

[9] http://java.sun.com/products/JavaManagement/
[10] http://java.sun.com/products/javabeans/

8.2 Connectors

The KAON SERVER comes with four system components, i.e. MBeans, that handle communication. First, there is an HTTP Adaptor from Sun that exposes all of the kernel's methods to a Web frontend. Additionally, we have developed Web Service (using the Simple Object Access Protocol) and RMI (Java Remote Method Invocation) connector MBeans. Both export the kernel's methods for remote access. Finally, we have developed a local connector that embeds the KAON SERVER into the client application.

For the client side there is a surrogate object called RemoteMBeanServer that implements the MBeanServer interface. It is the counterpart to one of the three connector MBeans mentioned above. Similar to the CORBA stubs, the application uses this object to interact with the MBeanServer and is relieved of all communication details. The developer can choose which of the three options (local, RMI, Web Service) shall be used by RemoteMBeanServer. In addition, surrogate objects may be developed that relay the communication to a specific MBean.

8.3 Interceptors

As explained in section 7, interceptors are software entities that look at a request and modify it before the request is sent to the component.

In the kernel, each MBean can be registered with an invoker and a stack of interceptors. A request received from the client is then delegated to the invoker first before it is relayed to the MBean. The invoker object is responsible for managing the interceptors and sending the requests down the chain of interceptors towards the MBean. For example, a logging interceptor can be activated to implement auditing of operation requests. An authorization interceptor can be used to check that the requesting client has sufficient access rights for the MBean.

Apart from security, invokers and interceptors are useful to achieve other goals. E.g., when a component is being restarted, an invoker could block and queue incoming requests until the component is once again available (or the received requests time out), or redirect the incoming requests to another MBean that is able to fulfill them.

8.4 Data APIs

The functionality described so far, i.e. the Management Core, Connectors and Interceptors could be used in any domain not just the Semantic Web. In the remaining subsections we want to highlight the specialties which make the KAON SERVER suitable for ontologies that follow Semantic Web language standards and the Semantic Web, in particular.

First, the KAON Tool suite has been made deployable. Two Semantic Web Data APIs for updates and queries are defined in the KAON framework - an

RDF API and an ontology data-oriented called KAON API. Their implementations result in functional components that are discussed in subsection 8.5. Furthermore, we are currently developing functional components that enable the semantic interoperation of Semantic Web ontologies (cf. section 4) as well as an Ontology Repository. Several external modules (inference engines in particular) are also deployable, as we have developed proxy components for them. All of them are discussed in the remaining subsections. Before talking about the API implementations and other functional components, the following paragraphs describe the APIs briefly.

RDF API

The RDF API consists of interfaces for the transactional manipulation of RDF models with the possibility of modularization, a streaming-mode RDF parser and an RDF serializer for writing RDF models. The API features object oriented representations of entities defined in [8] as interfaces. An RDF model consists of a set of statements. In turn, each statement is represented as a triple (subject, predicate, object) with the elements either being resources or literals. The corresponding interfaces feature methods for querying and updating those entities, respectively.

Ontology API

Our ontology data-oriented API, also known as KAON API, currently realizes the ontology language described in [22]. We have integrated means for ontology evolution and a transaction mechanism. The interface offers access to KAON ontologies and contains classes such as Concept, Property and Instance. The API decouples a client from actual ontology storage mechanisms.

8.5 Functional Components

The KAON API is implemented in different ways like depicted in figure 5. All of the implementations have been made deployable and are discussed subsequently in more detail. We also included descriptions of additional functional components, i.e. Ontology Repository, OntoLift, Semantic Interoperation and finally external modules.

RDF Mainmemory Implementation

This implementation of the RDF API is primarily useful for accessing in-memory RDF models. That means, an RDF model is loaded into memory from an XML serialization on startup. After that, statements can be added, changed and deleted, all encapsulated in a transaction if preferred. Finally, the in-memory RDF model has to be serialized again in order to make changes persistent.

RDF Server

The RDF Server is an implementation of the RDF API that enables persistent storage and management of RDF models. It uses a relational database whose physical structure corresponds to the RDF model. Data is represented using four tables, one represents models and the other one represents statements contained in the model. The RDF Server uses a relational DBMS and relies on the JBoss Application Server[11] that handles the communication between client and DBMS.

KAON API on RDF API

As depicted in figure 5, implementations of the ontological KAON API may use implementations of the RDF API. E.g., the KAON API can be realized using the mainmemory implementation of the RDF API for transient access and modification of a KAON ontology.

Engineering Server

A separate implementation of the KAON API can be used for ontology engineering. This implementation provides efficient implementation of operations that are common during ontology engineering, such as concept adding and removal by applying transactions. A storage structure that is based on storing information on a metamodel level is applied here. A fixed set of relations is used, which corresponds to the structure of the used ontology language. Then individual concepts and properties are represented via tuples in the appropriate relation created for the respective meta-model element. This structure is well-suited for ontology engineering, where the number of instances (all represented in one table) is rather small, but the number of classes and properties dominate. Here, creation and deletion of classes and properties can be realized within transactions.

Integration Engine

Another implementation of the KAON API is currently under development which lifts existing databases to the ontology level. To achieve this, one must specify a set of mappings from some relational schema to the chosen ontology, according to principles described in [23]. E.g. it is possible to say that tuples of some relation make up a set of instances of some concept, and to map foreign key relationships into instance relationships.

Ontology Repository

One optional component currently developed is a Ontology Repository, allowing access and reuse of ontologies that are used throughout the Semantic Web, such as WordNet for example. Within the WonderWeb project several of them have been developed [4].

[11] http://www.jboss.org

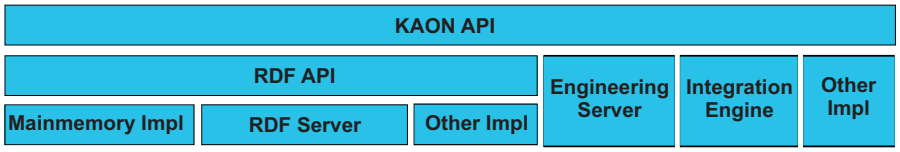

Fig. 5. KAON API Implementations

OntoLift

Another component realized in WonderWeb is OntoLift aiming at leveraging existing schema structures as a starting point for developing ontologies for the Semantic Web. Methods have been developed for deriving ontology structures for existing information systems, such as XML-DTD, XML-Schema, relational database schemata or UML specifications of object-oriented software systems. The Lift tool semi-automatically extracts ontologies from such legacy resources.

Semantic Interoperation

A functional component already developed, realizes the OWL Lite language on top of a SQL-99 compliant database system [3]. In addition, several others will later allow semantic interoperation between different types of ontology languages as a response to the requirement put forward in section 4. We already mentioned RDFS, OWL Lite, OWL DL and OWL Full in the introduction. Besides, there are older formats, like DAML+OIL and also proprietary ones like KAON ontologies [22]. It should be possible to load KAON ontologies into other editors, like OILEd [11], for instance. Information will be lost during ontology transformation as the semantic expressiveness of the respective ontology languages differ.

External Modules

External modules live outside the KAON SERVER. Proxy components are deployed and relay communication. Thus, from a client perspective, an external module cannot be distinguished from an actual functional component. At the moment we are adapting several external modules: Sesame [16], Ontobroker [6] as well as a proxy component for description logic classifiers that conform to the DIG interface[12], like FaCT [15] or Racer[7].

9 Related Work

Several systems approach some ideas relevant for an Application Server for the Semantic Web. However, all of them focus on RDF(S) or on ontology

[12] Description Logic Implementation Group, http://dl.kr.org/dig/

languages not specific to the Semantic Web and cannot be extended very easily.

RDFSuite [10] is provided by ICS-Forth, Greece with a suite of tools for RDF management, among those is the so-call RDF Schema specific Database (RSSDB) that allows storing and querying RDF using RQL. For the implementation of persistence an object-relational DBMS is exploited. It uses a storage scheme that has been optimized for querying instances of RDFS-based ontologies. The database structure is tuned towards a particular ontology structure.

Sesame [16] is a RDF Schema-based repository and querying facility developed by Aidministrator Nederland bv as part of the European IST project On-To-Knowledge. The system provides a repository and query engine for RDF data and RDFS-based ontologies. It uses a variant of the RDF Query Language (RQL) that captures further functionality from RDFSchema specification when compared to the RDFSuite RQL language. Sesame shares its fundamental storage design with RDFSuite.

Stanford Research Institute's OKBC (Open Knowledge Base Connectivity) is a protocol for accessing knowledges bases (KBs) stored in Knowledge Representation Systems (KRSs) [2]. The goal of OKBC is to serve as an interface to many different KRSs, for example, an object-oriented database. OKBC provides a set of operations for a generic interface to underlying KRSs. The interface layer allows an application some independence from the idiosyncrasies of specific KRS software and enables the development of generic tools (e.g., graphical browsers and editors) that operate on many KRSs.

The Ontolingua ontology development environment [1] provides a suite of ontology authoring tools and a library of modular, reusable ontologies. The tools in Ontolingua are oriented toward the authoring of ontologies by assembling and extending ontologies obtained from a library.

Developed by the Hewlett-Packard Research, UK, Jena [18] is a collection of RDF tools including a persistent storage component and a RDF query language (RDQL). For persistence, the Berkley DB embedded database or any JDBC-compliant database may be used. Jena abstracts from storage in a similar way as the KAON APIs. However, no transactional updating facilities are provided.

Research on middleware circles around so-called service oriented architectures (SOA)[13], which are similar to our architecture, since functionality is broken into components - so-called Web Services - and their localization is realized via a centralized replicating registry (UDDI)[14]. However, here all components are stand-alone processes and are not manageable by a centralized kernel. The statements for SOAs also holds for previously proposed distributed

[13] http://archive.devx.com/xml/articles/sm100901/sidebar1.asp
[14] http://www.uddi.org/

object architectures with registries such as CORBA Trading Services [12] or JINI[15].

Several of today's application servers share our design of constructing a server instance via separately manageable components, e.g. the HP Application Server[16] or JBoss[17]. Both have the Microkernel in common but follow their own architecture which is different from the one presented in our paper. JBoss wraps services like databases, Servlet and Enterprise JavaBeans containers or Java Messaging as components. HP applies its CSF (Core Services Framework) that provides registry, logging, security, loader, configuration facilities. However, whether JBoss nor HP AS deliver ontology-based registries, association management nor are they suitable for the Semantic Web, in particular.

10 Conclusion

This article presented the requirements and design of an Application Server for the Semantic Web as well as an implementation - the KAON SERVER. It is part of the open-source Karlsruhe Ontology and Semantic Web Tool suite (KAON). From our perspective, the KAON SERVER will be an important step in putting the Semantic Web into practice. Based on our experiences with building Semantic Web applications we conclude that such a server will be a crucial cornerstone to bring together so far disjoint components.

KAON SERVER still is work in progress. We are currently developing aforementioned functional components like the Ontology Repository, Semantic Interoperation, OntoLift. The Web Service connector will be enhanced by semantic descriptions whose source is the registry. In the future, we envision to integrate means for information quality - a field of research that deals with the specification and computation of quality criteria. Users will then be able to query information based on criteria like "fitness for use", "meets information consumers needs", or "previous user satisfaction" [5]. We will also support aggregated quality values, which can be composed of multiple criteria.

Acknowledgements

The KAON project (cf. http://kaon.semanticweb.org) is a meta-project carried out at the Institute AIFB, University of Karlsruhe and the Research Center for Information Technologies (FZI), Karlsruhe. KAON consolidates the results obtained in several government-funded projects and fulfills requirements imposed by industry projects carried out at the FZI. Individual software within KAON is developed within different projects. Development for KAON SERVER is funded by the EU-FET project WonderWeb (cf. http://www.wonderweb.org).

[15] http://www.jini.org
[16] http://www.bluestone.com
[17] http://www.jboss.org

References

1. Fikes, R., Farquhar, A., Rice, J., Tools for Assembling Modular Ontologies in Ontolingua., In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, AAAI Press / The MIT Press, 1997, pp. 436-441, ISBN 0-262-51095-2
2. Chaudhri, V., Farquhar, A., Fikes, R., Karp, P., Rice, J., OKBC: A Programmatic Foundation for Knowledge Base Interoperability.,In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, AAAI Press / The MIT Press, 1998, pp. 600-607, ISBN 0-262-51098-7
3. Grosof, B., Horrocks, I., Volz, R., Decker, S., Description Logic Programs: Combining Logic Programs with Description Logic, In *Proceedings 12th International World Wide Web Conference (WWW12), Semantic Web Track, 2003, Budapest, Hungary.*
4. Oltramari, A., Gangemi, A., Guarino, N., and Masolo, C., DOLCE: a Descriptive Ontology for Linguistic and Cognitive Engineering (preliminary report), In *Proceedings 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW2002), Siguenza, Spain (in press).*
5. Felix Naumann, *Quality-driven query answering for integrated information systems*, Lecture Notes in Computer Science, vol. 2261, Springer, 02 2002.
6. Decker, S., Erdmann, M., Fensel, D., Studer, R. Ontobroker: Ontology based access to distributed and semi-structured information, In *Database Semantics - Semantic Issues in Multimedia Systems, IFIP Conference Proceedings*, Kluwer, volume 138, pages 351-369, 1998
7. Haarslev, V., Moeller, R., RACER System Description., In *Proceedings of Automated Reasoning, First International Joint Conference, IJCAR (Rajeev Goré and Alexander Leitsch and Tobias Nipkow, eds.)*, vol. 2083, Lecture Notes in Computer Science, Springer, 2001, pp. 701-706.
8. O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax specification. Internet: http://www.w3.org/TR/REC-rdf-syntax/, 1999.
9. Dan Brickley and R. V. Guha. Resource description framework (RDF) schema specification 1.0. Internet: http://www.w3.org/TR/2000/CR-rdf-schema-20000372/, 2000.
10. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ics-forth rdfsuite: Managing voluminous rdf description bases. In *2nd International Workshop on the Semantic Web (SemWeb'01), in conjunction with Tenth International World Wide Web Conference (WWW10), Hongkong, May 1, 2001*, pages 1–13, 2001.
11. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. Oiled: a reasonable ontology editor for the semantic web. In *Proc. of the Joint German Austrian Conference on AI, number 2174 in Lecture Notes In Artificial Intelligence, pages 396-408*. Springer, 2001.
12. Juergen Boldt. Corbaservices specification, 3 1997.
13. E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho, A. Maedche, B. Motik, D. Oberle, C. Schmitz, S. Staab, L. Stojanovic, N. Stojanovic, R. Studer,

G. Stumme, Y. Sure, J. Tane, R. Volz, and V. Zacharias. KAON - towards a large scale semantic web. In *Proceedings of EC-Web 2002*. Springer, 2002.
14. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*, volume 1. John Wiley and Son Ltd, 1996.
15. I. Horrocks. The fact system. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*. Springer, 1998.
16. Frank van Harmelen Jeen Broekstra, Arjohn Kampman. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Proceedings International Semantic Web Conference 2002*. Springer, 2002.
17. Michel Klein, Atanas Kiryakov, Damyan Ognyanov, and Dieter Fensel. Ontology versioning and change detection on the web. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02), Sigenza, Spain, October 1-4, 2002*, 2002.
18. Brian McBride. Jena: Implementing the RDF model and syntax specification. In *Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001, Hongkong, China, May 1, 2001*, 2001.
19. Deborah L. McGuinness. Proposed compliance level 1 for WebOnt's ontology language OWL. Knowledge Systems Laboratory, Stanford University.
20. Sergej Melnik. RDF API. Current revision 2001-01-19.
21. Stefan Decker Michael Sintek. Triple - an RDF query, inference, and transformation language. In *Proceedings ISWC'2002*. Springer, 2002.
22. B. Motik, A. Maedche, and R. Volz. A conceptual modeling approach for building semantics-driven enterprise applications. In *Proceedings of the First International Conference on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, November 2002.
23. L. Stojanovic N. Stojanovic, R. Volz. A reverse engineering approach for migrating data-intensive web sites to the semantic web. In *IIP-2002, August 25-30, 2002, Montreal, Canada (Part of the IFIP World Computer Congress WCC2002)*, 2002.
24. L. Stojanovic, N. Stojanovic, and S. Handschuh. Evolution of metadata in ontology-based knowledge management systems. In *1st German Workshop on Experience Management: Sharing Experiences about the Sharing of Experience, Berlin, March 7-8, 2002, Proceedings*, 2002.
25. L. Stojanovic, N. Stojanovic, and R. Volz. Migrating data-intensive web sites into the semantic web. In *ACM Symposium on Applied Computing SAC 2002*, 2002.
26. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. Ontoedit: Collaborative ontology development for the semantic web. In *Proceedings of the 1st International Semantic Web Conference (ISWC2002), June 9-12th, 2002, Sardinia, Italia*. Springer, 2002.
27. Raphael Volz, Daniel Oberle, and Rudi Studer. Views for light-weight web ontologies. In *Proceedings of the ACM Symposium on Applied Computing SAC 2003, March 9-12, 2003, Melbourne, Florida, USA*, 2003.