Lifecycle-Support in Architectures for Ontology-Based Information Systems

Thanh Tran[1] and Peter Haase[1] and Holger Lewen[1] and Óscar Muñoz-García[2] and Asunción Gómez-Pérez[2] and Rudi Studer[1]

[1] Institute AIFB, Universität Karlsruhe, Germany
{dtr,pha,hle,rst}@aifb.uni-karlsruhe.de
[2] Universidad Politécnica de Madrid, Spain {omunoz,asun}@fi.upm.es

Abstract. Ontology-based applications play an increasingly important role in the public and corporate Semantic Web. While today there exist a range of tools and technologies to support specific ontology engineering and management activities, architectural design guidelines for building ontology-based applications are missing. In this paper, we present an architecture for ontology-based applications—covering the complete ontology-lifecycle—that is intended to support software engineers in designing and developing ontology-based applications. We illustrate the use of the architecture in a concrete case study using the NeOn toolkit as one implementation of the architecture.

1 Introduction

Ontology-based applications play an increasingly important role in the public and corporate Semantic Web. Major companies like Oracle[3] and IBM[4] have invested in semantic technologies. These efforts and work from the research community have led to a number of concrete implementations to support specific ontology engineering and management activities. Yet, there are not many ontology-based information systems (OIS) available that can exploit these technologies to deliver added value for the end user.

Partly, this is due to the lack of guidance for software engineers to develop OIS. Methodologies for the development of knowledge-based applications (e.g. CommonKADS [1]) can be applied to OIS, but normally focus purely on knowledge engineering. Architectures for semantic web services involve ontologies, but naturally focus on services. For instance, WSMO [2] or ODE-SWS [3] provide ontology-based mechanisms to formally describe services. While ontologies are a main component of these frameworks, ontology management features are not supported. Also guidance as to how ontologies can be used and managed at runtime by the service platform are not provided. Even results of the W3C group on best practices and deployment[5] cover only usage scenarios and guideline for ontology developments. The Semantic Web Framework (SMF) proposal [4] focussofes on identifying and describing components including their dependencies. This work is complementary to our work in the sense that while SMF so far

[3] http://www.oracle.com/technology/tech/semantic_technologies/index.html
[4] http://www.alphaworks.ibm.com/tech/semanticstk
[5] http://www.w3.org/2001/sw/BestPractices/

identifies and classifies components required for managing ontology, we focus on OIS architectures with lifecycle-support.

In particular, we discuss activities that must be supported in OIS on the basis of the notion of ontology lifecycle. Applying best practices and architecture paradigms such as SOA [5] and J2EE [6] from the software engineering community, we develop a generic architecture of integrated OIS that can even support scenarios where usage and engineering activities are intertwined at runtime. This architecture aims to provide a guideline for software engineers to design OIS. As an implementation of this architecture, we also discuss the NeOn toolkit[6], which provides a concrete framework containing reusable components that can be leveraged for the implementation of OIS. We demonstrate the application of both the architecture and the NeOn toolkit on the basis of a case study in the fishery domain.

The paper is structured as follows: In Section 2, we start with the discussion of the ontology lifecycle in ontology-based information systems. In Section 3, we present the generic architecture for OIS and illustrate how this architecture supports ontology lifecycle management. In Section 4, we then provide an instantiation of the generic architecture using the NeOn toolkit within a case study in the fishery domain. We conclude the paper with an outlook in Section 5.

2 Lifecycle Management of Ontologies

In this section, we briefly present existing work on the *ontology lifecycle*. The concept has mainly been used in methodologies for ontology engineering [7]. In the following, we give a compiled overview of these methodologies to present a simple lifecycle model (see Fig. 1). This model considers not only the engineering, but also the usage of ontologies at runtime as well as the interplay between usage and engineering activities.

2.1 Ontology Engineering

While the individual methodologies for ontology engineering vary, they agree on the main lifecycle activities, namely *requirement analysis*, *development*, *evaluation*, and *maintenance*, plus orthogonal activities such as project management. In the following, we focus on the first three engineering-related activities as described in the literature and then discuss maintenance in the context of usage-related activities.

Requirement Analysis: In this step, domain experts and ontology engineers analyze scenarios, use cases, and, in particular, intended retrieval and reasoning tasks performed on the ontology.

Development: This is the step in which the methodologies vary most. We therefore present an aggregated view on the different proposals for ontology development.

The initial step is the identification of already available reusable ontologies and other sources such as taxonomies or database schemas. Once reusable ontologies are found, they have to be adapted to the specific requirements of the application. This may include both backward (understanding, restructuring, modifying) and forward (modifying, extending) engineering of these reusable ontologies w.r.t. some design patterns.

[6] `http://www.neon-toolkit.org/`

Then, the ontologies are translated to the target representation language. Because of the expressivity-scalability tradeoff involved in reasoning, it may be desirable to tweak the degree of axiomatization, e.g. for performance. An important aspect in development is collaboration. Existing proposals for reaching consensus knowledge involve the assignment of roles and the definition of interaction protocols for knowledge engineers.

Integration: Inspired by the componentization of software, recent approaches advocate the modularization of ontologies [8]. Accordingly, the result of the development step shall be a set of modularized ontologies rather than one monolithic ontology. These modules have to be integrated, e.g. via the definition of import declarations and alignment rules. This integration concerns not only the modules that have been developed for the given use case. For interoperability with external applications, they may be embedded in a larger context, e.g. integrated with ontologies employed by other OIS.

Evaluation: Similar to bugs in software, inconsistencies in ontologies impede their proper use. So the initial evaluation step is to check for inconsistencies, both at the level of modules and in an integrated setting. Furthermore, ontologies also have to be assessed w.r.t. specific requirements derived from the use cases. Note that any deficiencies detected in this phase have to be addressed, i.e. led back to development.

2.2 Ontology Usage

Usage encompasses all activities performed with an ontology after it has been engineered. So far, the lifecycle as described in the literature is more of a static nature, just like the software lifecycle. Namely, if all requirements are met, the ontology will be deployed and the lifecycle continues with ontology evolution—also referred to as maintenance in literature. In this phase, new requirements may arise which are fed back into the loop, e.g. incorporated into the next release, which is then redeployed. Current lifecycle models however do not incorporate activities involved in the actual usage of ontologies. We will elaborate on these activities and based on them, show that the lifecycle can be dynamic.

Search and **Retrieval** and **Reasoning:** Once the ontologies have been created, they can be used to realize information access in the application, for example via search and retrieval. Typically an OIS involves a reasoner to infer implicit knowledge. The schema can be combined with instance data to support advanced retrieval, e.g. schema knowl-

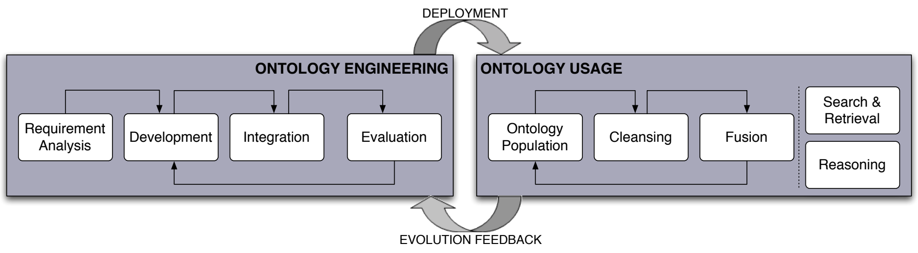

Fig. 1. Lifecycle Model

edge exploited for query enhancement (refinement, expansion), and A-Box reasoning to retrieve also inferred knowledge.

Note these are two generic exemplary tasks that shall illustrate the use of ontologies. In the actual application, search and retrieval may be only two of the many ontology-related operations that are embedded in more complex (business) logic implementing a concrete use case. These usages of ontologies may require support by the following application-independent lifecycle activities that are also performed at runtime:

Ontology Population: To populate the knowledge base (KB), instances may be collected from the user, e.g. via forms. A substantial overhead may be imposed to the user when all instance data has to be created manually. This burden can be alleviated by a (semi)-automatic population of the KB. Part of this knowledge creation step are also the manipulation and deletion of instances.

Cleansing and Fusion: Automatically extracted knowledge cannot be assumed to have the desired quality. Enhancing instance data may include identification and merging of conceptually identical instances that are only differently labeled (object identification) as well as fusion at the level of statements, e.g. merging redundant statements.

Both the population and the fusion steps may lead to inconsistencies which have to be resolved. Consider a user requesting data that yet has to be crawled from external sources. Then, inconsistencies that may arise in the process have to be resolved at runtime for the user to be able to continue his work. Found inconsistencies are fed back to debugging and the development-phase of the ontology lifecycle. That is, ontology evolution—the loop from usage back to engineering activities—is not only due to changing requirements but is also necessary for the runtime usage of ontologies.

3 A Generic OIS Architecture with Lifecycle Support

In this section, we present a generic architecture that aims to serve as a guideline for the development of any IS that involves ontologies. Hence, generic use cases that have to be considered may involve mere ontology engineering, mere ontology usage or a combination of both. Therefore, lifecycle activities discussed in the last section will be incorporated as functional requirements. Due to the possible dynamic nature of the lifecycle, it has to be supported in an integrated architecture that allows for a dynamic interaction of engineering and usage activities.

We will start with an overview and continue with a detailed elaboration on the components for lifecycle support. Then, we show how this generic architecture can be adopted for the development of OIS with concrete functional requirements. While the presented architecture abstracts from specific application settings, we also discuss how concrete architecture paradigms can be applied to meet technological requirements.

3.1 Overview of the Architecture

The proposed architecture as shown in Fig. 2 is organized in layers according to the control- and data flow (the control flow is indicated by the arrows) as well as the degree of abstraction of the constituent components. The former means that components at a higher layer invoke and request data from components at the lower layers. The latter means that components at the same abstraction level can be found on the same architecture layer. A single operation of components at a higher level of abstraction can trigger several low level operations. For example, a functionality provided by an ontology-based application front-end may invoke some ontology usage services, each of them, in turn, making use of several core ontology services. These services rely on requests to specific data sources, which are accessed via connectors of the data abstraction layer.

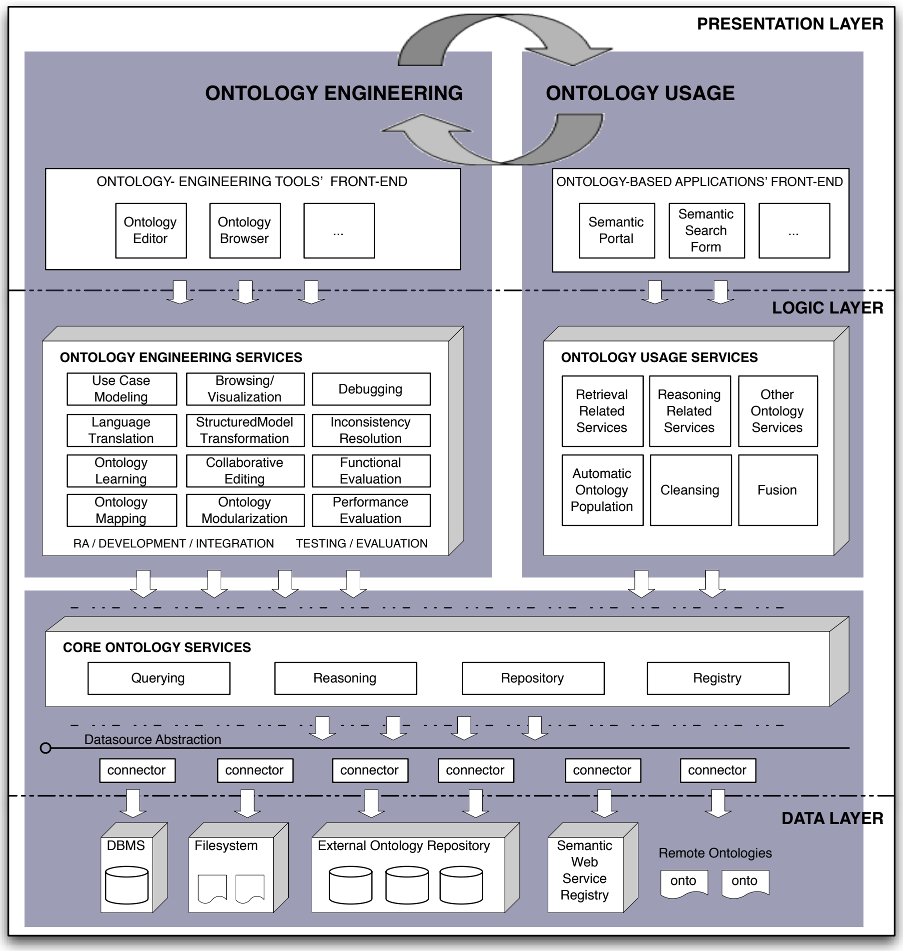

Fig. 2. The Generic Architecture

Note that many of the concepts employed for this architecture proposal, i.e. the presentation components, platform services, data source abstraction and connectors follow J2EE and SOA best practices. Also, the organization in (three different) layers is inspired from the n-tier architecture—a well-known organization principle in software engineering. We now briefly discuss these concepts and the components at the different layers (see [6, 5] for more information on J2EE and SOA best practices).

The Data Layer: This layer hosts any kind of *datasources*, including *databases* and *file systems*. This may also encompass ontological sources such as *external ontologies* hosted in repositories, *semantic web services* hosted in registries and any ontology on the web that can be retrieved via its URI. Note that services external to the system can be regarded as a component of the data layer because their processing is transparent to the internal components. The processing can be considered a black-box that simply provides data for internal components (see *connectors* in [9]).

The Logic Layer: At this layer, there are application-specific services that are implemented for a particular use case and operate on specific object models. The former encapsulate the processing logic and the latter capture the data. These services invoke *ontology lifecycle services* to manage and retrieve semantic data. Accordingly, object models may encapsulate data coming from conventional datasources like databases (data) or from ontological sources (semantic data), or both. In any case, the actual data comes from a persistent storage, most often a database. The data source abstraction can be used to hide specific datasource implementations by providing a uniform API and specific connectors. While not shown in Fig. 2, services at the logic layer run on a specific platform, which provides orthogonal functionalities for the management, configuration, and identification (registry) of services as well as access control and security.

The Presentation Layer: This layer hosts presentation components that the user interacts with. These components could be simply *pages* or *interactive forms* of a web-based system or more *sophisticated UIs* of a desktop application that contains a variety of widgets. The engineering and usage operations performed by the user on these components translate to calls to services situated at the logic layer. The data returned by these services is then presented by the components together with the static content.

We will now continue with a more detailed elaboration on ontology-related services.

3.2 Ontology-related Services

Ontology-related services are organized in one layer for core services and one layer for the higher level ontology lifecycle services. While the control and data flow of lifecycle and core services are top-down as shown in Fig. 2, the interaction between the different lifecycle activities typically corresponds to the structure of the corresponding lifecycle activities, e.g. they follow a sequential flow. However, the actual interaction depends on the needs of a particular use case. That is, ontology lifecycle services can be invoked and controlled by application-specific services as needed.

Core Ontology Services: Functionalities offered by services at this layers are used by lifecycle services. An *ontology registry service* is used to find and publish ontologies. An *ontology repository service* provides access, manipulation and storage (persistence is supported by the lower level datastore) at the level of ontologies and at the level of ontology elements. That is, repository functionalities are also available for axioms, concepts, properties, individuals etc. The repository service also includes logging and versioning to ensure reversibility. Besides the common repository retrieval methods, a *query service* offers a generic mechanism for retrieval. Finally, a *reasoning service* is available for standard reasoning tasks such as consistency checking, classification etc.

Ontology Engineering Services: The architecture contains services for the requirement analysis that has functionalities similar to the ones supported in an IDE for software development, e.g. for requirements elicitation, *modeling of use cases* and specification of reasoning and retrieval tasks involved in the use cases.

In the actual development, services are provided for ontology browsing, visualization, editing and integration. In particular, *browsing* and *visualization* supporting ontologies as well as non-ontological artifacts such as interface signatures, data base schema, and UML models to help in identifying reusable artifacts. To enable reuse, there are services for the *translation* of existing ontologies to the target representation formalism. Services for (semi)-automatic *transformation* of non-ontological sources to ontologies are also incorporated into the architecture [10] to facilitate reuse. This transformation is possible in both directions to ensure the interoperability of ontology data w.r.t. these data sources. Services for *ontology learning* are also provided to accelerate the development process by the generation of a base version that can be further refined. Implementations of specific interaction protocols enable a *collaborative editing* process. The *mapping service* includes support for the identification and specification of ontology modules as well as their relations and dependencies. Also, it includes the specification of concept mappings required for the alignment of ontologies.

After the base ontologies have been further developed, adapted to requirements and integrated, they have to be tested and evaluated. For these tasks, there are services for *debugging* (identification of axioms that are responsible for or affected by the inconsistency) and for the *inconsistency resolution* of the conflicts [11]. Also, there are services that evaluate the coverage of the ontology w.r.t. the representative set of retrieval and reasoning tasks envisaged for the use cases (*functional evaluation*). Finally, *performance evaluation* services are essential to meet the requirements and are incorporated into the architecture. In order to meet performance targets for particular scenarios, different configurations for ontology axiomatization may be considered.

Ontology Usage Services: In Fig. 2, some application-specific services are shown to illustrate that ontologies may be used as a technology to implement use cases of a particular OIS. This can involve *reasoning*, *retrieval*, but also *other tasks* enabled by ontologies. In order to support these ontology-based services, the architecture contains the following lifecycle usage services that are rather independent from specific use cases.

Services that can automatically *populate* the KB reduce the effort needed for the manual creation of instance data. These services are performed by agents that request external ontology data as well knowledge extractors that crawl external non-ontological sources. They implement learning algorithms to extract instances from text and multimedia contents. Some of these population services (and ontology learning services) may incorporate procedures for natural language processing [12] as subcomponents.

Finally, the quality of the acquired instance data has to be ensured. *Cleansing* services are available to adapt the format and labels to the application requirements. The same instances extracted from different sources may have different labels. Knowledge *fusion* services identify and resolve such occurrences. Similarly, knowledge acquired from different sources may be redundant and often contradictory. This is also addressed by the fusion services. These services may implement a semi-automatic process, which involves the user and engineering services such as debugging and editing. The arrows in Fig. 2 illustrate this interaction between usage and engineering services. It is provided by the evolution support, a feature part of aforementioned usage services, which possibly require interaction with engineering services.

3.3 Designing OIS with the Generic Architecture

We now discuss how this architecture can act as a reference that can be adapted to match functional and technological requirements of a particular OIS.

Matching Functional Requirements: The presented architecture is very generic and targets the management of the entire ontology lifecycle. Implementing the whole architecture would result in a fully-fledged integrated system that supports both the engineering and the application of ontologies. However, a particular application often requires only a subset of the envisaged services.

Applications may feature only engineering, or only usage of ontologies that already have been engineered using another system. Then only engineering and usage services, respectively, have to be incorporated into the concrete architecture of the final application. In general, the functional requirements of the system have to be analyzed. Then these requirements have to be mapped to services of the architecture. Finally, for each of the identified services, more fine-grained functionalities have to be derived w.r.t. the use cases to be supported by the application.

For instance, an application that only uses RDF(S) ontologies may not need any lifecycle services at all. Imagine a web application, which simply presents FOAF profiles manually imported from external sources. Then only core ontology services are needed to import, store and retrieve information from the profiles. A more sophisticated version may employ agents to crawl profiles from the web. Even then, only population and basic cleansing is needed, because due to the use of RDF(S), no inconsistencies can arise that would require engineering services. Now, imagine an application using OWL ontologies to manage resources of a digital library. Resources are annotated with ontology concepts that can be defined by the user. Most annotations are extracted automatically and even new concept descriptions are suggested by the system to capture the knowledge contained in new library resources. Clearly, this application would need a wide range of usage and engineering services and hence, an integrated application with lifecycle support.

Matching Technological Requirements: The presented architecture is of abstract nature and free of assumptions about specific technological settings. For the development of a specific application, it can be used as a reference to identify the components (as discussed previously) and to organize them with the suggested abstraction layers and control-flow. Then, given specific technological constraints, a concrete architecture paradigm can be chosen and applied to the abstract architecture. These paradigms capture best practices in different application settings and can also give additional guidance for OIS engineering. We will now outline standard paradigms in software engineering and discuss for which exemplary settings they are most appropriate.

Architecture paradigms can be distinguished along three dimensions, namely the degree of distribution, coupling and granularity of components. Distribution can range from non-distributed rich client, over client-server, three-tier [13], multi-tier to fully-distributed P2P architectures. The last two dimensions make up the differences of two more concrete architecture paradigms with specific platform assumptions, namely the component-oriented multi-tier J2EE architecture [6] and the Service-oriented Architecture (SOA) [5]. While J2EE comprises of tightly-coupled and relatively fine-grained components, SOA advocate the use of loosely-coupled and coarse-grained services.

The main idea behind multi-tier architectures is the encapsulation of each tier, meaning any tier can be upgraded or replaced without affecting the other tiers. While this organization principle has been adopted (where layer stands for tier), the proposed architecture does not make any assumptions about how components may be distributed. In fact, the layered organization can be seen as an orthogonal principle that can be combined with any of the mentioned paradigms.

For instance, elements of the architecture can be implemented as components of a desktop application, e.g. the backend maps to a file system, services and control-flow map to Plain Old Java Objects (POJOs) and their call hierarchy and GUI components map to Swing widgets. In another use case, more flexible access may be required, the application logic may call for more processing capabilities, and the amount of data cannot be managed efficiently by a file system. Then, a database can be employed as backend, data access can be provided by Data Access Objects (DAO) and lifecycle services are realized as Enterprise Java Beans (EJB) of a J2EE platform, and front-ends are implemented as Java Server Pages (JSP) to deliver contents over the web. In some cases lifecycle components could be tightly integrated with other internal systems via J2EE connectors [9]. In other cases external parties may want to choose from different offerings and therefore demand a more flexible way to discover ontology services at runtime and to interact with them on the basis of a standardized protocol. Here, SOA may be the choice: The fine-grained functionalities of some lifecycle components are encapsulated in form of coarse-grained services exposed to consumers via WSDL and SOAP. Instead of using a completely new SOA platform, one may go a more evolutionary way advocated by major J2EE vendors, i.e. switch to a Service Component Architecture (SCA)[7] that implements SOA. SCA provide guidelines for decoupling service implementation and service assembly from the details of underlying infrastructure capabilities. Components can then offer their functionalities as services that can also be consumed externally. However for internal consumption, they do not necessarily have to be loosely coupled—since tight coupling can avoid the overhead of creating, parsing and transporting messages over the network.

In all, the generic architecture gives guidelines for the identification and organization of components. The examples above illustrate that there are many other aspects that have to be considered given concrete requirements. After the choice for a concrete platform and the paradigm to be applied on the architecture, guidance can then be found in the respective reference architectures, e.g. see [9] for J2EE, [5] for SOA and SCA.

4 Case Study – An Instantiation of the Generic Architecture

In this section, we discuss the application of the generic architecture w.r.t. a concrete case study of the NeOn project[8] at FAO (United Nations Food and Agriculture Or-

[7] `http://www.osoa.org/display/Main/Service+Component+Architecture+Home`
[8] `http://www.neon-project.org/web-content/`

ganization). Within this case study, we are developing an ontology-based information system to facilitate the assessment of fisheries stock depletion by integrating the variety of information sources available. The FAO Fisheries department manages statistical data on fishing, GIS data, information on aquaculture, geographic entities, description of fish stocks, etc. Although much of the data are ‘structured’, they are not necessarily interoperable. In addition, there are information resources that are not available through databases but as parts of websites, or as documents, etc. These data sources could be better exploited by bringing together related and relevant information with the use of ontologies, to provide inference-based services, enabling policy makers and national governments to make informed decisions. In this context, the goal of the case study is to implement an ontology-based Fishery Stock Depletion Assessment System (FS-DAS) as well as an application to manage the fishery ontologies and their lifecycle. In the following, we follow the methodological guidelines of the previous section: We illustrate how to match the functional requirements by analyzing the use cases w.r.t. the individual phases of the ontology lifecycle, and finally show how we match technical requirements in the realization of these use cases as two particular configurations of the NeOn toolkit.

4.1 Uses Cases within the Lifecycle of the Fishery Ontology

In the ontology lifecycle of the case study we find a clear separation between the ontology engineering and ontology usage phases. In fact, we find two different sets of users that are involved in ontology engineering and ontology usage. We will now discuss selected use cases in the lifecycle that provide functional requirements that need to be covered in the architecture of the system.

Ontology Engineering: The ontology engineering environment needs to put mechanisms in place to allow all actors involved in the process to create and maintain distributed networked ontologies and ontology mappings in the fishery knowledge community. These mechanisms require many of the generic ontology engineering services discussed in the previous chapter.

There are several actors involved in the engineering phase of the fishery ontology lifecycle [14], including experts in ontology modeling, ontology editors, and subject matter experts. Each of the actors needs to be supported in different use cases of the ontology development. Ontology development follows a well defined collaborative workflow, which needs to be supported in the engineering environment. Further, contextualized visualization and editing modes—depending on the actor and the task to be performed—are important for the usability and effectiveness of the engineering environment.

Due to the scale and heterogeneity of the various information sources, there must be an easy way to create mappings between ontologies in a manual and semi-automatic way. Modularization of ontologies—i.e. creation of modules manually and semi-automatically as well as merging modules—must also be taken into account. The generation of ontologies from textual sources is another key issue. Given a textual corpora the system must provide a list of candidate elements of the ontology (classes, instances and relations between concepts), showing the documents and excerpts supporting the extracted terminology.

Before publishing a new version of an ontology, debugging and evaluation must be performed. This involves checking for logical consistency, making comparisons with other ontologies, and evaluating structural and functional properties of the ontology.

Ontology Usage: The fishery ontologies are used within the FSDAS system to assist the users—i.e. fishery experts in the FAO Fishery department—in gathering, analyzing and producing information on the status and trends of fish stock. For example, a fishery expert may want to research why the stock of tuna is depleting in the Mediterranean Sea. For this purpose, the ontology-based FSDAS allows authorized users to browse and query a knowledge base of fisheries digital resources.

The major use case within FSDAS is to perform ad-hoc queries, using both free-text and ontology elements such as concepts and relations, against the fishery data sources. In this context, matching of keywords and phrases to ontology elements, assistance on query formulation and query refinement are supported by the system. The second major use case related to information access is browsing and navigating the fishery data sources, e.g. using the ontology to find related data instances for a given concept, i.e. analyzing stock depletion. In interacting with the FSDAS, users are able to maintain a profile and store favorite queries.

At runtime, the FSDAS directly integrates dynamic resources to populate the fishery ontology with data instances, e.g. by crawling remote system websites, etc. Additionally, users have the ability to annotate data instances with the FSDAS system.

Finally, the users of the FSDAS system have the ability to comment the use of the application and in particular propose modifications to the ontology. This information is fed back into the ontology development phase to support the evolution of the fishery ontology. Once a new version of the fishery ontology has been approved, it can be deployed in the runtime FSDAS application, closing the loop of the ontology lifecycle.

4.2 Instantiation of the Generic Architecture

We now discuss how we address technical requirements in the realization of the case study using the NeOn toolkit [15] as an implementation of our generic architecture. The NeOn toolkit provides an infrastructure and software components for both the engineering and the runtime usage of ontologies. For the case study, two applications are developed, one for the engineering of the fishery ontologies and one for the FSDAS runtime system. They are used by different sets of end users—i.e. ontology engineers and fishery experts—with different technological requirements. They are realized as two particular configurations of the NeOn toolkit, i.e. realized using a subset of components provided by the toolkit. In particular, both applications are built on a shared infrastructure, shared data sources and shared core ontology services. However, whereas one is configured with a bundle of ontology engineering services, the other relies completely on usage services—as shown in Figure 3.

The *Core Ontology Services* shared across the engineering and runtime environments are based on the NeOn ontology model API—an API that supports the management of rule extended OWL ontologies and ontology mappings [16]—and include services for reasoning and querying, an ontology repository and registry etc. Based on the core ontology services, there are higher-level services that cover the ontology usage and engineering use cases discussed above.

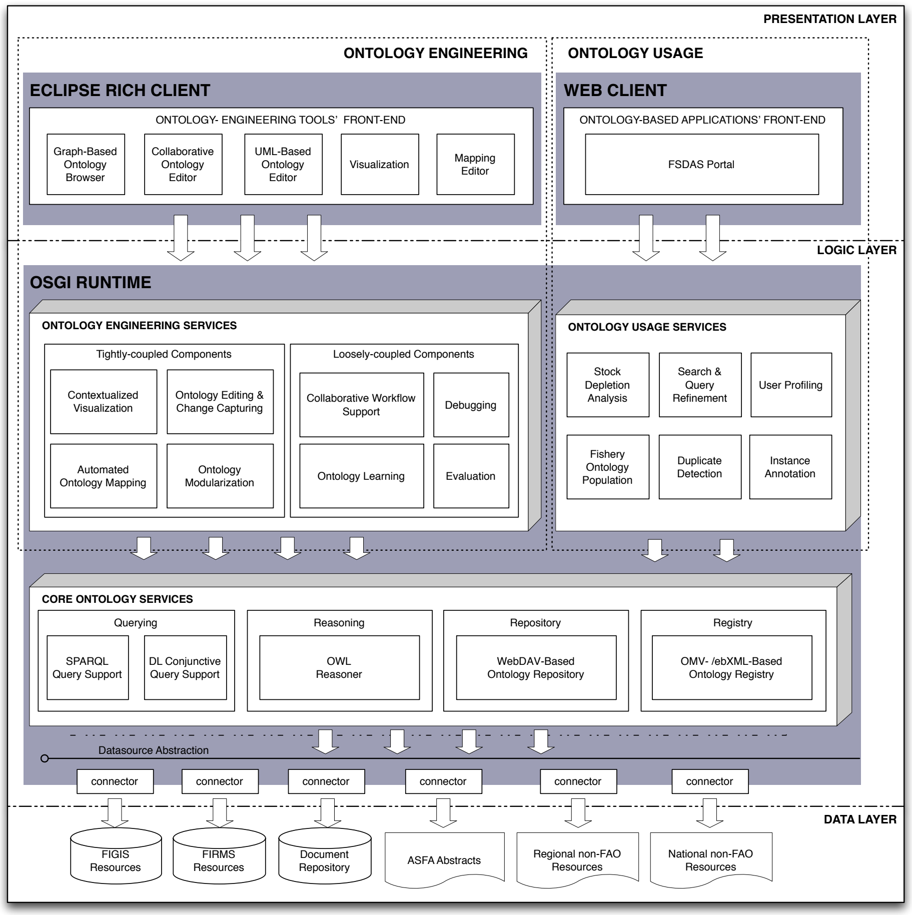
PRESENTATION LAYER
ONTOLOGY ENGINEERING
ONTOLOGY USAGE
ECLIPSE RICH CLIENT
ONTOLOGY- ENGINEERING TOOLS' FRONT-END
Graph-Based Ontology Browser
Collaborative Ontology Editor
UML-Based Ontology Editor
Visualization
Mapping Editor
WEB CLIENT
ONTOLOGY-BASED APPLICATIONS' FRONT-END
FSDAS Portal
LOGIC LAYER
OSGI RUNTIME
ONTOLOGY ENGINEERING SERVICES
Tightly-coupled Components
Contextualized Visualization
Ontology Editing & Change Capturing
Automated Ontology Mapping
Ontology Modularization
Loosely-coupled Components
Collaborative Workflow Support
Debugging
Ontology Learning
Evaluation
ONTOLOGY USAGE SERVICES
Stock Depletion Analysis
Search & Query Refinement
User Profiling
Fishery Ontology Population
Duplicate Detection
Instance Annotation
CORE ONTOLOGY SERVICES
Querying
SPARQL Query Support
DL Conjunctive Query Support
Reasoning
OWL Reasoner
Repository
WebDAV-Based Ontology Repository
Registry
OMV- /ebXML-Based Ontology Registry
Datasource Abstraction
connector
connector
connector
connector
connector
connector
DATA LAYER
FIGIS Resources
FIRMS Resources
Document Repository
ASFA Abstracts
Regional non-FAO Resources
National non-FAO Resources

Fig. 3. Case Study Architecture based on the NeOn Platform

The shared infrastructure of core services is based on OSGi, an open Java-based platform[9] that is the foundation of a service oriented architecture. Besides many other standard platform services, OSGi defines a lifecycle model and a service registry, that allows for the dynamic interaction of services. The standard implementation used within the NeOn toolkit is Eclipse Equinox[10]. The OSGi platform is designed to support both *distributed client-server configurations* and *desktop configurations* via Eclipse/OSGi. In fact, for the realization of the case study, we implement the ontology engineering environment using a desktop configuration, i.e. an Eclipse rich-client application, while the FSDAS is realized using a distributed configuration where user interfaces of the application are web-based. The reason for this lies in the technical requirements of the

[9] `http://www.osgi.org/`
[10] `http://eclipse.org/equinox/`

two different user groups, where the ontology engineers require a rich tool set on their local desktops, whereas the fishery experts as non-technical users want to work with light-weight applications in web environments.

In the ontology engineering environment, we distinguish between tightly and loosely coupled components: Loosely coupled components are non-interactive, large grain, potentially remotely used components. They are integrated as Web Services. In contrast, tightly coupled components are highly interactive, fine grained, locally used components. Every engineering use case discussed previously is addressed by a component consisting of one or more Eclipse plugins. The modularity via the plugin concept of Eclipse follows the philosophy that "everything is a plugin". Using this plugin concept, components for ontology editing, visualization and other functionalities for the different actors in the engineering process through the integration of various components provided by the NeOn toolkit. As an example, the collaborative ontology engineering is supported by a number of services that are provided by loosely and tightly coupled components: The local ontology editing is tightly integrated with the change capturing, while certain services to support the collaborative workflow such as conflict resolution as well as validation and evaluation services are loosely coupled as web services.

The FSDAS application is realized as a distributed configuration, which combines OSGi with server-based technology. In our configuration, this is achieved by embedding web-server technology in the OSGi runtime platform, making the ontology usage services accessible in the web-based FSDAS portal.

5 Conclusion

In order to provide guidance for the development of ontology-based information systems, we have developed an integrated architecture that takes the complete ontology lifecycle into account. As we have illustrated, such an architecture is required to address use cases where ontology usage and engineering are intertwined at runtime, resulting in a dynamic feedback loop. This loop and the lifecycle activities act as functional requirements, which are addressed in our proposal for a generic architecture for ontology-based information systems.

We have discussed how to adapt this architecture to functional requirements of specific use cases, from simple ontology applications to systems for integrated ontology engineering and management. To demonstrate the value of our architecture, we have shown its application in a concrete case study, using the NeOn toolkit. This toolkit is a concrete implementation of our generic architecture, integrating reusable lifecycle components from and for the community that can facilitate the adoption of semantic technologies. In the future, we will further add components to the toolkit as well as promote the integration of tools externally developed by the community, either as tightly-coupled components or loosely-coupled services. We expect the toolkit to evolve to a more complete set of reusable components which—combined with the generic architecture—can serve as guidelines for the design and can be leveraged for the implementation of many other OIS with lifecycle support.

Acknowledgements

This work was partially supported by the European Commission in the IST projects NeOn (IST-2006-027595) and X-Media (IST-FP6-026978).

References

1. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W., Wielinga, B.: Knowledge Engineering and Management: The CommonKADS Methodology. MITPress (1999)
2. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A Semantic Service-Oriented Architecture. In: Proceedings of the IEEE International Conference on Web Services (ICWS'05), IEEE Computer Society Washington, DC, USA (2005) 321–328
3. Gómez-Pérez, A., González-Cabero, R., Lama, M.: ODE SWS: A Framework for Designing and Composing Semantic Web Services. IEEE Intelligent Systems **19** (2004) 24–31
4. García-Castro, R., Suárez-Figueroa, M.C., Gómez-Pérez, A., Maynard, D., Costache, S., Palma, R., Euzenat, J., Lécué, F., Léger, A., Vitvar, T., Zaremba, M., Zyskowski, D., Kaczmarek, M., Dzbor, M., Hartmann, J., Dasiopoulou, S.: Architecture of the Semantic Web Framework. Technical Report D1.2.4, Knowledge Web Consortium (2007)
5. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: OASIS Reference Model for Service Oriented Architecture v1.0, OASIS Official Committee Specification, approved August 2006 (2006)
6. Singh, I., Stearns, B., Johnson, M., Enterprise Team: Designing Enterprise Applications with the J2EE (tm) Platform. The Java Series. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2002)
7. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering. Advanced Information and Knowlege Processing. Springer (2003)
8. Wang, Y., Bao, J., Haase, P., Qi, G.: Evaluating formalisms for modular ontologies in distributed information systems. In: Proc. of The First International Conference on Web Reasoning and Rule Systems (RR2007). LNCS 4524, Springer (2007) 178–182
9. Sharma, R., Stearns, B., Ng, T.: J2EE (tm) Connector Architecture and Enterprise Application Integration. The Java Series. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2002)
10. Wu, Z., Chen, H., Wang, H., Wang, Y., Mao, Y., Tang, J., Zhou, C.: Dartgrid: a semantic web toolkit for integrating heterogeneous relational databases. In: Semantic Web Challenge at 4th International Semantic Web Conference (ISWC'06), Athens, USA (2006)
11. Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y.: A framework for handling inconsistency in changing ontologies. In: Proceedings of the Fourth International Semantic Web Conference (ISWC2005). Volume 3729 of LNCS., Springer (2005) 353–367
12. Valarakos, A., Paliouras, G., Karkaletsis, V., Vouros, G.: Enhancing Ontological Knowledge through Ontology Population and Enrichment. Proc. of the 14th Int. Conference on Knowledge Engineering and Knowledge Management (EKAW 2004), LNAI **3257** (2004) 144–156
13. Eckerson, W.W., et al.: Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. Open Information Systems **10** (1995)
14. Iglesias, M., Caracciolo, C., Jaques, Y.: NeOn Deliverable D7.1.1 Specification of user requirements in the fishery case study . Technical Report D7.1.1, NeOn Consortium (2007)
15. Waterfeld, W., Weiten, M., Haase, P.: NeOn Deliverable D6.2.1 Specification of NeOn reference architecture and NeOn APIs. Technical Report D6.2.1, NeOn Consortium (2007)
16. Haase, P., Rudolph, S., Wang, Y., Brockmans, S., Palma, R., Euzenat, J., d'Aquin, M.: Networked Ontology Model. Technical Report D1.1.1, NeOn Consortium (2006)