A Reference Architecture for Semantic Content Management Systems

Fabian Christ, Benjamin Nagel

s-lab - Software Quality Lab
University of Paderborn
Warburger Str. 100
D-33098 Paderborn
fchrist@s-lab.upb.de
bnagel@s-lab.upb.de

Abstract: Content Management Systems (CMS) lack the ability of managing semantic information that is part of the content stored in a CMS. On the other hand, a lot of research has been done in the field of Information Extraction (IE) and Information Retrieval (IR), respectively. Additionally, the vision of the Semantic Web yields to new software components that make semantic technology usable for application developers. In this paper, we combine IE/IR concepts with the technologies of the Semantic Web and propose a new family of CMS, called Semantic CMS (SCMS), with advanced semantic capabilities. We provide a reference architecture for SCMS and prove its value along two implementations. One implementation was created as part of the Interactive Knowledge Stack research project and another one in a one-year student project exploring the design of an SCMS for the software engineering domain.

1 Introduction

The growing importance of an efficient content management is omnipresent in nearly each aspect of daily business. The challenges posed by the raising amount of content are well-known [AL99] and are becoming greater due to the high availability of content from various sources like the Internet. Through this, more and more unstructured content needs to be handled and therefore aggravate the management of content by Content Management Systems (CMS).

Addressing these challenges, the "Semantic Web Wave" [LHL01] brought up several technologies that enable the definition of semantics as machine-readable metadata, termed "semantic metadata". Recent approaches [Sar08], like named-entity recognition, clustering and classification use the foundations and facilitate the automatic enrichment of content with semantic metadata. The semantic metadata allow the automatic structuring

This work has been part-funded by the European Commission under grant agreement FP7-ICT-2007-3/ No. 231527 (IKS - Interactive Knowledge Stack).

135

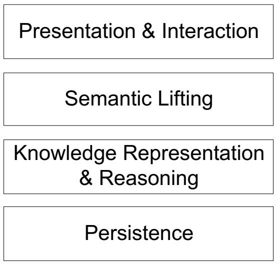
Presentation & Interaction
Semantic Lifting
Knowledge Representation & Reasoning
Persistence

Figure 1: The four layers of an SCMS

and thereby the further processing like searching or filtering. In addition, these metadata enable the reasoning of new knowledge based on the content.

Summarizing the research trends of the last years, several technological foundations and algorithms have been developed to face the challenges of efficient content and knowledge management. On the other hand, only little attention has been paid to the integration of these approaches from a software engineering perspective. The integration of semantic functionalities has significant impact on the architecture. The CMS architecture has to be extended in comparison to traditional CMS, because the storage and processing of semantic metadata in a semantic CMS (SCMS) needs to be considered. Furthermore, concepts for reasoning and knowledge extraction need to be integrated and the controlling of their processing must be handled.

The authors participate in the Interactive Knowledge Stack (IKS)[1] project, that is focused on building an open and flexible technology platform for SCMS. As a working hypothesis, when starting the project in 2009, we thought of four layers, shown in Figure 1, that would be required to have in an SCMS architecture. The top layer called *Presentation & Interaction* for presenting knowledge to the end user and support a direct interaction with this knowledge. To extract knowledge from content the second layer called *Semantic Lifting* is used. In this layer, a given content is lifted to semantic level by extracting knowledge from it. To generate new knowledge based on existing knowledge and to represent this knowledge within an SMCS we need the third layer, called *Knowledge Representation and Reasoning*. The last layer persists the new knowledge in the *Persistence* layer.

Starting with the initial four layer architecture we began our research on requirements. We also implemented semantic components in an iterative development process. The requirements were gathered by asking industrial CMS vendors for their needs and by doing research on possible semantic features [CES$^+$09]. The requirements were consolidated and implemented in prototypical semantic components in an agile development process. The architecture was refined and adapted with each iteration of research and development [CEN$^+$10].

In this paper, we present the resulting reference architecture for SCMS that addresses the

[1]http://www.iks-project.eu (July 27, 2011)

domain-specific characteristics of such systems. This reference architecture has been implemented in two software development projects that are described as case studies. The reference architecture can be adapted in two different ways. On the one hand, the architecture can be used to enhance a traditional CMS with semantic features. On the other hand, the reference architecture provides support for the specification of SCMS from scratch.

The remainder of this paper is structured as follows. At first, an overview about the state-of-the-art in engineering CMS is given in Section 2. In Section 3, traditional CMS architectures and their characteristics are discussed. Section 4 introduces the SCMS reference architecture that is applied in two case studies described in Section 5. Finally, the presented results are concluded and an outlook about future work is given in Section 6.

2 Related Work

Recent research provides only few reference models for the development of SCMS. Based on the technologies and foundations developed for the Semantic Web, a stack architecture, termed the "Semantic Web Stack" as described in [Ber00], has been designed. This architecture has been extended in [HPPH05] with a rule layer. Both approaches provide a conceptual architecture by defining layers and assigning appropriate semantic technologies to these layers. Though, both approaches do not provide architectures that can be operationalized in the sense of development of software. They neither specify functionalities provided by layers or components nor the interaction between the layers.

In [BKvH02] Sesame is introduced, a generic architecture model that focuses on storing and querying metadata in RDF and RDF schema. The architecture includes functional components that can be deployed. As described, this approach is restricted on a specific technology and thus limited to RDF and RDF schema. It mainly addresses persistence and data access components, whereas other functionalities like knowledge extraction or user interaction are not considered sufficiently.

A server architecture for XML document management is proposed in [AMFK$^+$00]. The architecture consists of a set of high-level layers that focus on the processing of textual documents. The approach is limited to XML-based content and does not provide solutions for the storing or extraction of knowledge from existing content.

The "Context Broker Architecture" (CoBrA) described in [CFJ03] uses semantic web technologies in an architecture for context-aware information systems. The storage and access of knowledge and context reasoning is explicitly considered. The architecture is designed on a high level of abstraction and does not describe the different components and their interfaces in detail. In addition, dynamic processing of knowledge extraction is not in the focus of the approach.

Summarizing the state-of-the-art in architecturing of SCMS, existing approaches do not provide a generic model that is independent from concrete technologies and considers all aspects of knowledge extraction and management. Another shortcoming of existing architectures is the lack of user interaction with knowledge providing an actual value of semantic technologies for the end user.

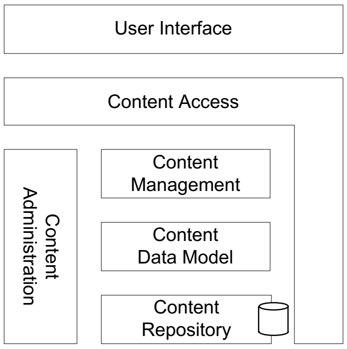

Figure 2: CMS Server Architecture

3 Traditional CMS Architecture

CMS architectures are built upon the concept of a 3-tier architecture with client, CMS server, and database backend. Figure 2 shows the internal server architecture of a CMS. The main difference of CMSs compared to other information systems is to focus on flexible content modeling and storage. Content data models as the representation of content and their persistence need to be highly adaptable to any domain or customer scenario.

A CMS User Interface at the top layer in Figure 2 presents the content and offers editorial features to create, modify, and manage content within its lifecycle. Access to the content itself is provided by a Content Access layer. This layer is used by the User Interface to get access to the content and the content management features of the CMS. Additionally, the Content Access layer can be used by third party software that may want to integrate the CMS into other applications.

The core management features are implemented in the Content Management layer. This layer provides functionalities for the definition of the domain or application specific Content Data Model. Access control and content lifecycle definitions are further typical management features implemented in this layer. The Content Data Model layer is conceptually placed below the Content Management layer that has the necessary features to manipulate the model. The Content Data Model is the application specific model based on the underlying Content Repository. The Content Repository defines the fundamental concepts and persistence mechanisms for any Content Data Model that is defined on top. The Content Management features are tightly related to the Content Administration layer to administer the CMS stack.

4 Reference Architecture

In an SCMS the content is stored in a traditional Content Repository and the knowledge about that content is additionally stored in a Knowledge Repository. Our proposal of an

138

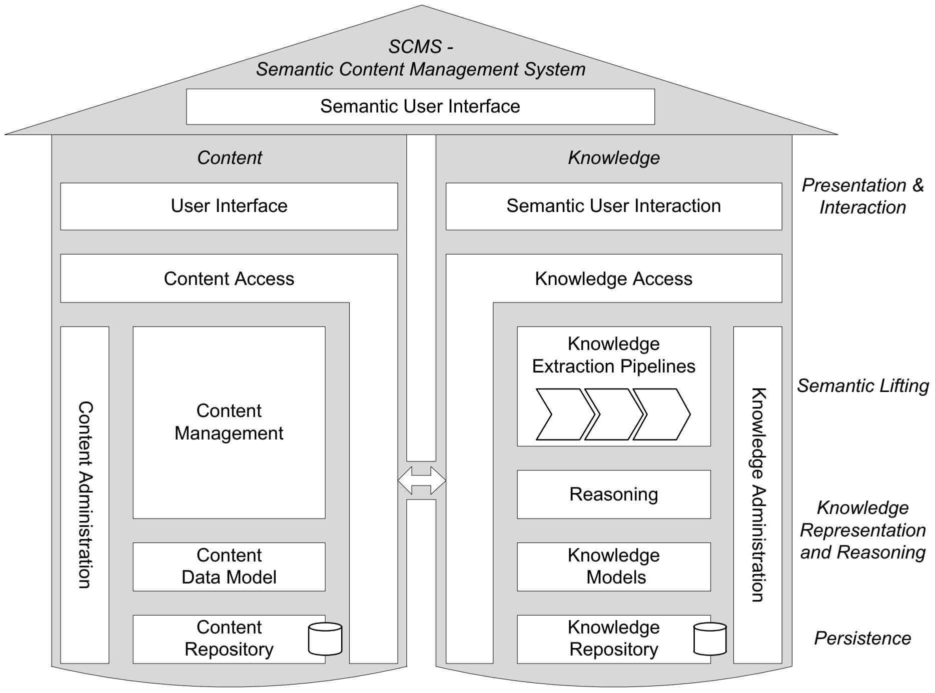

Figure 3: SCMS Reference Architecture

SCMS reference architecture, as shown in Figure 3, is designed in such a way that any existing CMS, that has an architecture similar to the one depicted in Figure 2, can be extended to become an SCMS. This ensures that a CMS can be semantified without any major changes to the existing CMS. To create an SCMS out of a CMS the traditional CMS content column is extended by a second knowledge column for the semantic features in parallel.

To interact with each other both columns are connected at the level of Content Access and Knowledge Access. The content of an SCMS is stored in the content column. Traditional content repositories are best prepared for this task. The content is transported from the content column to the knowledge column at the Content/Knowledge Access layer. For a loosely coupled solution, this could be done via some service implementation, e.g. RESTful Web Services [Fie00]. The delivered content can be analyzed by the components of the knowledge column. The obtained knowledge is stored in the knowledge column. Once the content column needs additional knowledge regarding some content it can query the content column. On the other hand the knowledge column can search for new or changed content in the content column.

In summary, a SCMS is a CMS with the capability of interacting with, extracting, managing, and storing semantic metadata about content. In the following, we will describe the SCMS reference architecture along our coarse grained four layer concept shown in

139

Figure 1 that consists of *Presentation & Interaction*, *Semantic Lifting*, *Knowledge Representation & Reasoning* and *Persistence*.

Presentation & Interaction In a traditional CMS, the user is able to edit and consume content through a user interface. When dealing with knowledge in SCMS we need an additional layer at the user interface level that allows a user to interact with content, called Semantic User Interaction. For example, a user writes an article and the SCMS recognizes the name of a person in that article. An SCMS includes a reference to an object representing that person – not only the person's name. The user can interact with the person object and see, e.g. its birthday. The person is a knowledge object that is part of a Semantic User Interaction. Access to knowledge is encapsulated through a Knowledge Access layer similar to the Content Access layer. Whereas the content column of Figure 3 provides access to the content from the User Interface, provides the knowledge column access to knowledge for Semantic User Interaction. By combining existing features of a CMS User Interface layer with new feature from the Semantic User Interaction layer, an SCMS provides a Semantic User Interface layer on top of both.

Semantic Lifting One problem for traditional CMS is the missing ability to extract knowledge in terms of semantic metadata from the stored content. Therefore, an SCMS defines a layer for Knowledge Extraction Pipelines that encapsulate algorithms for semantic metadata extraction. Typically, knowledge extraction is a multistage process [FL04] by applying algorithms known from the research field of Information Extraction and Retrieval. A Knowledge Extraction Pipeline defines that each stage of the pipeline produces results that are an additional input for the next stage [ELB07]. For example, a typical step in a Knowledge Extraction Pipeline is to identify all entities in a given content.

Knowledge Representation & Reasoning After lifting content to a semantic level this extracted information may be used as inputs for reasoning techniques in the Reasoning layer. Logical reasoning is a well-known artificial intelligence technique that uses semantic relations to retrieve knowledge about the content that was not explicitly known before.

To handle knowledge within the system we use Knowledge (representation) Models that define the semantic metadata used to express knowledge. These metadata are often defined along some ontology that specifies so-called concepts and their semantic relations. For example, persons and organizations are concepts and a semantic relation between these concepts may define that persons can be employees of organizations. Using this definition, one can express that a concrete person is an employee of an organization and this knowledge may have been extracted from a given content through a Knowledge Extraction Pipeline.

In the same way the content column provides an extra orthogonal layer for Content Administration there is a need for a corresponding construct to administer knowledge. Knowledge Administration leads from the management of Semantic User Interaction templates, over Knowledge Extraction Pipeline and Reasoning management to the administration of Knowledge Models and Repositories.

Persistence Knowledge is stored in a Knowledge Repository that defines the fundamental data structure for knowledge. State-of-the-art knowledge repositories implement a triple store where a triple is formed by a subject, a predicate, and an object. Influenced by the ideas of the semantic web a triple can be used to express any relation between a subject and an object. To make this a semantic relation one has to define the Knowledge Models on top of the Knowledge Repository, e.g. to specify the semantic meaning of a certain predicate.

In the following section, two case studies are introduced that demonstrate the applicability of our reference architecture in concrete project contexts.

5 Case Studies

The concepts introduced by the SCMS reference architecture are validated through two distinct implementation projects. The first project is the IKS reference implementation (IKS-RI) project with the goal to deliver an implementation of the Knowledge column of the presented SCMS architecture in Section 4. The IKS-RI is validated through an industrial early adopters program where CMS vendors are invited to integrate the IKS-RI technology into their existing CMS and by this they create an SCMS. The major part of the IKS-RI is implemented as its own open source sub-project hosted at the Apache Software Foundation. This IKS subproject is called Apache Stanbol and was founded to create an independent open source community around the presented concepts of an SCMS. In this paper we reflect on the current work in progress status plus the planned features of the IKS-RI. The IKS-RI 1.0 release is scheduled for the end of 2011 and the Apache Stanbol community is supposed to push the development further on after 2011.

The second project is a one-year student project that lasted from April 2010 to April 2011. The project was dedicated to develop an SCMS for the software engineering domain and is called Information-Driven Software Engineering – abbreviated as ID|SE. The idea is to create an SCMS that can handle large unstructured software specification documents and helps to extract the semantic information that is hidden in these documents. Therefore, the ID|SE platform tries to, e.g., identify pieces of content within a specification that describe a requirement.

5.1 IKS Reference Implementation

The IKS-RI[2] is used as a proof of concept implementation of the SCMS reference architecture. The implementation of IKS-RI and the development of the SCMS reference architecture is an intertwined process driven by open-source developers for the implementation and researchers for the reference architecture. Distinguished roles and viewpoints on the problem to support the development of SCMS ensured that all decisions made on

[2]http://code.google.com/p/iks-project/ (July 27, 2011)

141

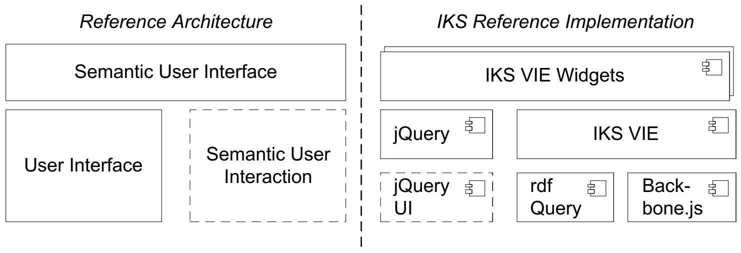

Figure 4: Reference Architecture compared to VIE Architecture

the implementation or the conceptual design were critically questioned.

The IKS-RI implements the Knowledge column of the SCMS reference architecture (see Figure 3). Its implementation is divided into different open-source projects. The IKS-RI developers, amongst others the authors of this paper, use and support several existing open-source projects for their work. Examples are the Apache Clerezza[3] project to manipulate semantically linked (RDF) data or the Apache OpenNLP[4] project for natural language processing. Additionally, the developers are actively involved in the development of a newly founded open-source project, called Apache Stanbol[5], to implement the required knowledge features. Before describing the work done at Apache Stanbol we will first focus on the Semantic User Interaction layer of the reference architecture.

Today, most CMS are web-based CMS whereas clients are primarily web browsers. Thus, the User Interface is implemented using modern Java Script browser technology. Along the traditional User Interface, the new Semantic User Interaction features are also required to work inside web browsers. The IKS/VIE[6] sub-project makes use of a set of Java Script libraries, namely jQuery, Backbone.js, and rdfQuery and creates new semantic interaction widgets on top. The VIE architecture is depicted in Figure 4 in comparison to the reference architecture. VIE provides a framework at the Semantic User Interaction layer for user interaction widgets that can be used to create user interface components in the Semantic User Interface layer. Such widgets realize user interaction with content plus knowledge. A simple scenario is to edit a text and insert not just the literal name of the person but instead the entity person with the associated knowledge about that person.

The Apache Stanbol architecture is built upon the OSGi [All11] component model implemented by the Apache Felix[7] project. The OSGi model supports elegant separation of different components required by the Knowledge column. Each component is based on a Resource Oriented Architecture [Ove07] exposing their interfaces in terms of a REST [Fie00] API. The aggregation of the components' interfaces forms an implementation of the Knowledge Access layer in Figure 3. This means that all components of Apache Stanbol address functionality within this layer and below of the reference architecture. Apache Stanbol

[3]http://incubator.apache.org/clerezza/ (July 27, 2011)
[4]http://incubator.apache.org/opennlp/ (July 27, 2011)
[5]http://incubator.apache.org/stanbol/ (July 27, 2011)
[6]http://github.com/IKS/VIE (July 27, 2011)
[7]http://felix.apache.org/ (July 27, 2011)

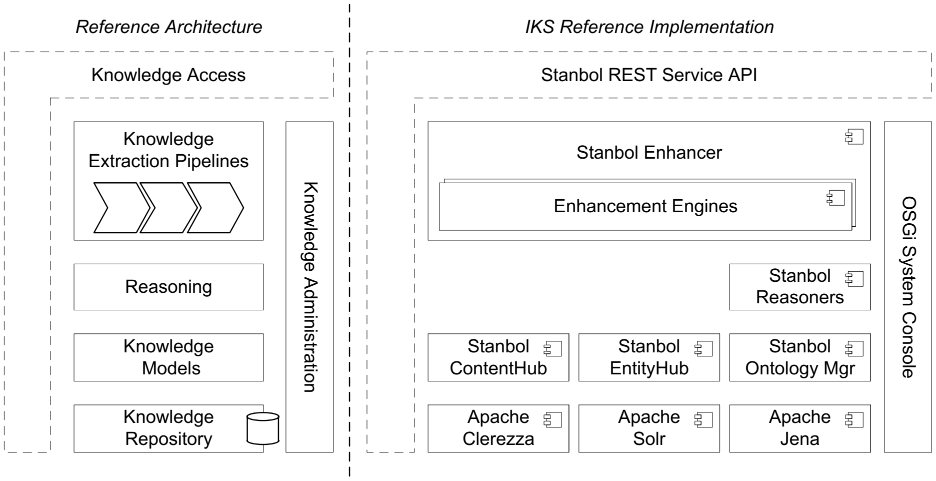

Figure 5: Reference Architecture compared to IKS-RI Architecture

does not address the top layers. The IKS-RI architecture for the SCMS layers starting with Knowledge Access is depicted in Figure 5.

Apache Stanbol defines a sub-framework, called Stanbol Enhancer, to implement Knowledge Extraction Pipelines. Such a pipeline consists of Enhancement Engines in which each engine is responsible to automatically extract one piece of information. Subsequent engines can use information extracted by previously executed engines. At the end of this process a given content is enhanced with the extracted metadata that become knowledge. The most important knowledge that can be extracted with Apache Stanbol is the identification of certain types of entities within the content. Supported entities are e.g. persons and locations. The extracted knowledge is represented in a triple-graph structure provided by Apache Clerezza and stored with the use of the Stanbol ContentHub. The ContentHub provides access to previously enhanced content and allows to retrieve already stored knowledge for a given content.

The EntityHub is used to retrieve semantic information about entities available through accessible Linked Open Data sources. A prominent example of a public Linked Open Data source is the DBPedia[8] archive which provides access to the data available through Wikipedia[9]. The EntityHub is able to search for entities in such data sources, to cache the information in an Apache Solr database, and to publish this information within Stanbol.

Reasoning about knowledge is prepared through the Stanbol Reasoner component which is able to integrate existing reasoning engines. This reasoning implementation requires knowledge that is structured according to an ontology that can be managed by the Stanbol Ontology Manager. The Ontology Manager provides a management API for ontologies expressed in the Web Ontology Language (OWL) [W3C09]. Clients can query the Stanbol

[8]http://dbpedia.org/ (July 27, 2011)

[9]http://www.wikipedia.org/ (July 27, 2011)

Reasoner to gain new knowledge that is only implicitly present in the stored knowledge. The Stanbol Reasoner processes existing knowledge represented in an OWL ontology and is able to retrieve new knowledge, e.g. by using the semantic relations between entities defined through the ontology. Ontologies can be stored using Apache Jena which supports semantic web standards like the used OWL.

The administration of all Apache Stanbol components is done via the OSGi System Console that is provided by the underlying Apache Felix OSGi implementation. Each component defines its own configuration parameters that can be manipulated via the System Console. For example, the used Linked Open Data source for the EntityHub is configurable through this mechanism.

5.2 Information-Driven Software Engineering

The ID|SE project[10] was a one-year student project with the goal to create an SCMS for the software engineering domain. Imagine a CMS in which one can upload unstructured requirements and specification documents and the system is able to identify specified requirements, actors, use cases, and components of the system plus the ability to reason about relations, e.g. between actors and use cases. Such an SCMS would help to structure the large amount of information written in plain text specifications. The ID|SE project used the publicly available specification of the German health card system for evaluation. The specification[11] consits of more than 10,000 pages of text written in German including some tables plus a few informal figures and is available in Word or PDF format.

In this paper, we focus on the ID|SE architecture and leave out the discussion about semantic features of the ID|SE platform. In Figure 6, we compare the reference architecture with the ID|SE version of an SCMS architecture. The ID|SE project realizes the Content column in Figure 3 on the basis of OpenCMS[12], an open-source CMS. In consequence, the ID|SE User Interface is implemented as an OpenCMS module. The focus of the ID|SE project was on features to analyze a given software specification document. The creation of sophisticated Semantic User Interaction components was not part of the project. That is the reason why the ID|SE architecture does not reflect Semantic User Interaction but only provides a User Interface without extra semantic capabilities.

The ID|SE Service Platform API provides access through web services to the platform services that are realized by the IE/IR Service Orchestrators. To connect the Knowledge column with the Content column provided by OpenCMS the ID|SE project implemented an adapter in OpenCMS that informs the ID|SE platform about newly uploaded content to trigger the semantic lifting machinery and to transfer the content from OpenCMS to the ID|SE platform.

The core of the ID|SE platform is a set of semantic lifting components that implement features from the research field of Information Extraction (IE) and Information Retrieval

[10]http://is.uni-paderborn.de/fachgebiete/fg-engels/lehre/ss10/pg-idse/pg-idse.html (July 27, 2011)
[11]http://www.gematik.de/ (July 27, 2011)
[12]http://www.opencms.org/ (July 27, 2011)

144

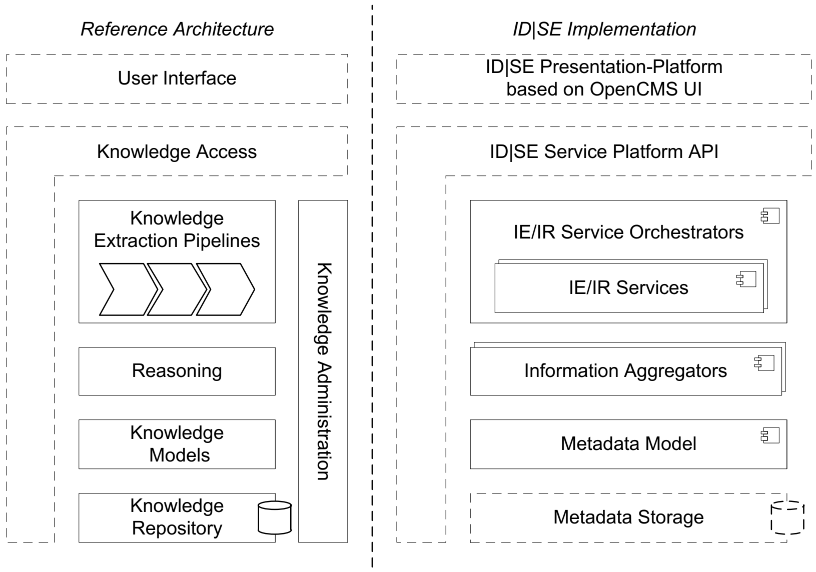

Figure 6: Reference Architecture compared to ID|SE Architecture

(IR), respectively. This IE/IR services are executed in a pipeline by the IE/IR Service Orchestrators. In the following, we present a list of the IE/IR services used by the platform to give an impression of how this Knowledge Extraction Pipeline depicted in Figure 7 works.

- *Content Extraction*
  Features to convert a given content in DOC or PDF format into a format that is suitable for being further processed by the pipeline.

- *Preprocessors*
  Features to pre-process the content by applying techniques like tokenization of text, sentence detection, part-of-speech tagging, and word stemming.

- *Classifier*
  A set of algorithms to classify pre-processed content according some classification schema. It is used for example to classify content as a text containing requirements or as a use case description.

- *Clusterer*
  Features to partition large sets of documents into clusters of similar documents. This is used, e.g., to identify similar requirements as they are partitioned into the same cluster with these techniques.

- *Named Entity Recognizer*
  Features to recognize named entities like actors, components, and use cases within the content.

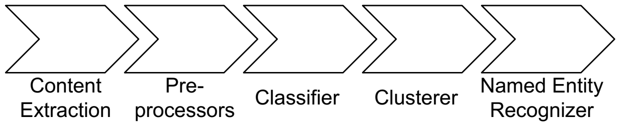

Figure 7: ID|SE IE/IR Services Pipeline

After processing the documents by the IE/IR Services, the ID|SE platform provides additional features that are aligned to the reasoning capabilities of the reference architecture, called Information Aggregators. The Information Aggregators layer consists of components that collect semantic metadata that were extracted by the IE/IR Services and generates new metadata on that basis. The ID|SE platform provides three aggregators.

- *Entity Cover Calculator*
  The Entity Cover Calculator computes the ratio between the coverage of named entities between two documents. This feature can be used, e.g., to search for documents that may be duplicates of each other like duplicate requirements.
- *Faceted Data Creator*
  The Faceted Data Creator generates metadata for a faceted search user interface component. A faceted search lets users browse through large sets of documents by selecting different document categories. Each selection of a new category reduces the number of matching documents and allows an easy way to find a searched document.
- *Use-Case Model Recognizer*
  Searches for use cases and actors in documents and creates UML [OMG10] use-case diagrams based on this information.

The knowledge extracted by the IE/IR Services and the Information Aggregators is represented using the ID|SE specific Metadata Model. This model defines that each document consists of a number of artifacts in which each artifact is stored as a character sequence. This character sequence can be annotated with metadata defining that a certain sub-sequence contains a specific semantic information. Additionally, character sequences form tokens and sentences. The Metadata Model is implemented using the Java Persistence API (JPA) [Gro06] for storing these data into the Metadata Storage. The ID|SE platform uses the open-source relational database MySQL[13] as its Metadata Storage which implements the Knowledge Repository layer of the reference architecture.

[13]http://www.mysql.com/ (July 27, 2011)

146

6 Discussion & Future Work

In this paper, we have motivated the need for a new generation of CMS that improve the handling of the growing amount of content by using semantic web technologies and IE/IR concepts. For these systems, termed Semantic Content Management Systems (SCMS) we proposed a reference architecture that supports the development of such systems by considering domain-specific requirements like the pipeline processing of knowledge extraction technologies. The reference model is intended to be used in different ways. The architecture can be adapted to develop an SCMS from scratch or to enhance existing, traditional CMS with semantic functionalities.

The applicability of our approach in these two different ways has been evaluated by the implementation of the reference architecture in two software development projects. Applying the reference model to concrete implementation projects, the architecture provided a starting point for integrating semantic web and IE/IR technologies. The proposed structure ensures the consideration of all relevant aspects of semantic content and knowledge management by appropriate concepts. Hereby, the presented approach attains benefits for the design and implementation of SCMS.

Addressing the experiences from the performed case studies, we identified three directions for the further work on our reference architecture. As a first step, the reference architecture will be evaluated as part of an overall evaluation in the IKS project. The applicability is measured by the expertises from the project consortium including partners from seven research institutions and six industrial CMS providers.

Secondly, we will use best practices and feedback given by architects and developers from our case studies to concretize the different layers and their functionality to fine-granular components. In addition, conceptual interfaces for the interaction between these components will be defined. Through this, the applicability of the reference architecture in the technical design will be improved.

Finally, we still see the potential for further improvement in the presentation and interaction layer of our architecture. The investigation of useful concepts for semantic user interaction and corresponding user interfaces is still part of our current work. The impact of these concepts on an architecture level needs to be analyzed in order to identify alignments for the reference architecture.

References

[AL99] Maryam Alavi and Dorothy E. Leidner. Knowledge management systems: issues, challenges, and benefits. *Commun. AIS*, 1999.

[All11] OSGi Alliance. OSGi Service Platform - Core Service Specification Version 4.3, 2011. http://www.osgi.org/Release4/HomePage (July 27, 2011).

[AMFK^{+}00] Tim Arnold-Moore, Michael Fuller, Alan Kent, Ron Sacks-Davis, and Neil Sharman. Architecture of a Content Management Server for XML Document Applications. In

147

1st International Conference on Web Information Systems Engineering (WISE 2000), 2000.

[Ber00] Tim Berners-Lee. Semantic Web - XML2000, 2000. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html (July 27, 2011).

[BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the first Int'l Semantic Web Conference (ISWC 2002)*, Lecture Notes in Computer Science, pages 54–68. Springer, 2002.

[CEN$^+$10] Fabian Christ, Gregor Engels, Benjamin Nagel, Stefan Sauer, Sebastian Germesin, Enrico Daga, and Ozgur Kilic. IKS Alpha Development. Deliverable, 2010. http://www.iks-project.eu/resources/iks-alpha-development (July 27, 2011).

[CES$^+$09] Fabian Christ, Gregor Engels, Stefan Sauer, Gokce B. Laleci, Erdem Alpay, Tuncay Namli, Ali Anil Sinaci, and Fulya Tuncer. Requirements Specification for the Horizontal Industrial Case. Deliverable, 2009. http://www.iks-project.eu/resources/requirements-capture-through-use-cases (July 27, 2011).

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. Semantic Web in a Pervasive Context-Aware Architecture. *IN ARTIFICIAL INTELLIGENCE IN MOBILE SYSTEM 2003 (AIMS 2003), IN CONJUCTION WITH UBICOMP*, pages 33–40, 2003.

[ELB07] Michael Thomas Egner, Markus Lorch, and Edd Biddle. UIMA GRID: Distributed Large-scale Text Analysis. In *Seventh IEEE International Symposium on Cluster Computing and the Grid, CCGRID 2007.*, pages 317–326, 2007.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm (July 27, 2011).

[FL04] David Angelo Ferrucci and Adam Lally. Building an example application with the unstructured information management architecture. *IBM Systems Journal*, pages 455–475, 2004.

[Gro06] EJB 3.0 Expert Group. JSR 220: Enterprise JavaBeans,Version 3.0 Java Persistence API, 2006. http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html (July 27, 2011).

[HPPH05] Ian Horrocks, Bijan Parsia, Peter Patel-Schneider, and James Hendler. Semantic Web Architecture: Stack or Two Towers? In *Principles and Practice of Semantic Web Reasoning*, pages 37–41. Springer, 2005.

[LHL01] Berners Lee, J Hendler, and O Lassila. The Semantic Web. *Scientific American*, 2001.

[OMG10] OMG. Unified Modeling Language (OMG UML), Superstructure, V2.3, 2010. http://www.omg.org/spec/UML/2.3/Superstructure/PDF (July 27, 2011).

[Ove07] Hagen Overdick. The Resource-Oriented Architecture. *IEEE Congress on Services*, pages 340–347, 2007.

[Sar08] Sunita Sarawagi. Information Extraction. *Foundations and Trends in Databases*, pages 261–377, 2008.

[W3C09] W3C. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2009. http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/ (July 27, 2011).

148