# A semantically Enriched Persistence Mechanism for Interactive Knowledge Stack[1]

Asuman DOGAC[1], Gokce B. LALECI[1], Gunes ALUC[2], Ali Anil SINACI[1] , Wernher BEHRENDT[3], Bertrand DELACRETAZ[4], Jean-Michel PITTET[4],
[1]*Software Research, Development and Consultancy Ltd.,*
*METU-KOSGEB Tekmer, Ankara, 06531, Turkey*
*Tel: +90 312 2102076, Fax: + 90 312 2105572, (asuman,gokce, anil)@srdc.com.tr*
[2]*Dept. of Computer Engineering,*
*Middle East Technical University, İnönü Bulvari, Ankara, 06531, Turkey*
*Tel: +90 312 2405598, Fax: + 90 312 2101259, (gunes)@srdc.metu.edu.tr*
[3] *Salzburg Research Forschungsgesellschaft m.b.H,*
*Jakob-Haringer Strasse 5/II, 5020 Salzburg, Austria*
Tel: +43 (0)662 2288 409 , Fax: +43 (0)662 2288 222,
*wernher.behrendt@salzburgresearch.at*
[4]*Day Software AG,*
*Barfüsserplatz 6, 4001 Basel, Switzerland*
*Tel: +41 61 226 98 98, Fax: +41 61 226 98 97*
*(bertrand.delacretaz, jean-michel.pittet)@day.com*

**Abstract:** Currently Content Management System or Knowledge Management System providers use a variety of persistency mechanisms including relational/object oriented databases and file management systems, or dedicated content repositories. None of these systems support semantics adequately, it is not possible to annotate the content objects through ontologies, or semantically search them by exploiting ontological reasoning. In the IKS Project, we are providing a semantically enriched data access and persistency mechanism for the Content Repositories to extract the implicitly expressed semantics in the Content Repositories, synchronizing this semantics with a Persistency Store that supports reasoning and rule engines. In this way it becomes possible to deploy horizontal semantic applications exploiting this hidden semantics such as semantic indexing and semantic search.

## 1. Introduction

There are hundreds of European SMEs developing Content Management System (CMS) and Knowledge Management System (KMS) solutions. These companies currently use either the "LAMP" stack (i.e. systems build on the basis of Linux, Apache, mySQL, and php) or technology stacks based on J2EE and .NET, as their development environment. These technologies connect databases and web-based user interfaces via web-servers, but they do not provide sufficient flexibility which is often needed to present the right information to the right audience using the right means of interaction.

[1] The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 231527

Although semantic web technologies promise declarative programming and rule-based inference in order to introduce the needed flexibility into software solutions, it can be said that the so-called Semantic Web "cake" is not a proper technology stack. Furthermore, Semantic Web applications have little genuine support for classical functions of content management. It is especially difficult for SMEs to orient themselves in midst of the many web-related standards to enrich their CMS/KMS development tools.

IKS Project (IST-231527 Interactive Knowledge Stack for small to medium CMS/KMS providers) will lead to a technology platform for CMS/KMS providers, which is much more semantically enabled and yet, much more manageable for developers, than the current set of technologies available for content and knowledge management. The Interactive Knowledge Technology Stack (Figure 1) to be developed will enable many open source CMS frameworks to become semantically enabled.

This technology stack consists of the following layers:

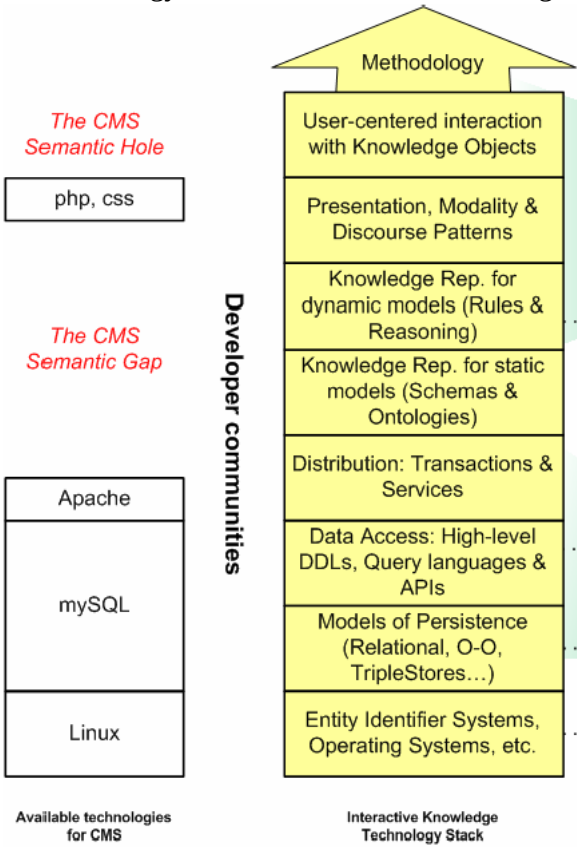



*Figure 1IKS Interactive Knowledge Stack*

**User centered interaction with knowledge objects:** This layer enables us to represent content objects as "computational objects" so that the user can interact with the knowledge instead of just consuming it.

**Presentation, modality and discourse patterns:** This layer provides explicit models of presentation, modality and of discourse between systems and users to facilitate interacting with knowledge objects.

**Knowledge representation for dynamic models - rules and reasoning:** This layer enables the definition of processes which are governed by constraints, usually expressed as rules which guard evaluation (reasoning).

**Knowledge representation for static models (schemas and ontologies):** This layer enables the definition of domain knowledge as ontologies.

**Distribution, Transactions and Services:** This layer enables the federation of heterogeneous data sources dealing with transactions, sessions and with the distribution of data.

**Data Access, High-level DDLs, Query languages & APIs:** This layer enables access to data sources ranging from relational databases to knowledge bases through a range of tools and languages from simple http requests to SQL and SPARQL queries.

**Data Model of Persistence Layer:** This layer enables the use of relational/object oriented databases in conjunction with RDF triple stores, enabling transactional support and reasoning on the data being stored by these persistence mechanisms.

Note that IKS does not address the bottom layer of operating systems and web-based entity identifier systems.

In this paper we particularly address the semantically enriched Data Access and Persistence Layers of this technology Stack. In Section 2, we present the objectives and technical architecture of the “Data Access and Persistence Layer”. Section 3 discusses potential benefits and conclusions.

## 2. Objectives and Methodology

Currently Content Management System or Knowledge Management System providers use a variety of persistency mechanisms including relational/object oriented databases and file management systems, or dedicated content repositories. A Content Repository is a high-level information management system that is a superset of traditional data repositories, which implements 'content services' such as: author based versioning, full textual searching, fine grained access control, content categorization and content event monitoring [2]. With the growing number of proprietary content repositories, the need for a common, standardized access mechanism for content repositories has become apparent. The JCR API [3] aims to provide such an interface as a standard, implementation independent way to access content bi-directionally on a granular level within a content repository. CMIS [4] is a standards proposal to achieve interoperation between content management systems by defining a set of Web services that can be used across disparate content repositories.

The databases or the content repositories that are currently being used by CMS/KMS systems do not support reasoning, as a result the CMS/KMS providers need to implement the business logic inside the applications itself instead of declaratively representing the domain knowledge through ontologies and rules, and exploiting description logic or rule based reasoners.

On the other hand, as semantic persistence mechanisms, the triple stores have been implemented as purpose-built databases for the storage and retrieval of Resource Description Framework (RDF) metadata such as Jena[5], Sesame[6], and RDFSuite [7]. These usually support reasoning through plug-in reasoners, and sometimes through rule engines. However they usually do not provide conventional database functionalities such as transactional access and concurrency control. The functionalities and the storage paradigms of conventional databases, content repositories and triple stores are not compatible at all. This makes it nearly impossible - even for research projects - to find a coherent system architecture for data storage, rule representation and reasoning over content instances and schemas. On top of this, there is an impedance mismatch between available semantic persistence mechanisms and IKS objectives. The aim of such triple stores is to provide a persistence mechanism to the already existing semantic knowledge represented as ontologies and rules. However the objective of IKS data access and persistence layer is to provide mechanisms to semantically represent data that is being kept usually in databases and content repositories and reason on them.

To address these problems, bi-directional adapters between conventional persistence mechanisms and knowledge bases are being developed to enable the connection between middle and upper layer of the IKS stack with the lower, persistence layer of traditional media and content management systems.

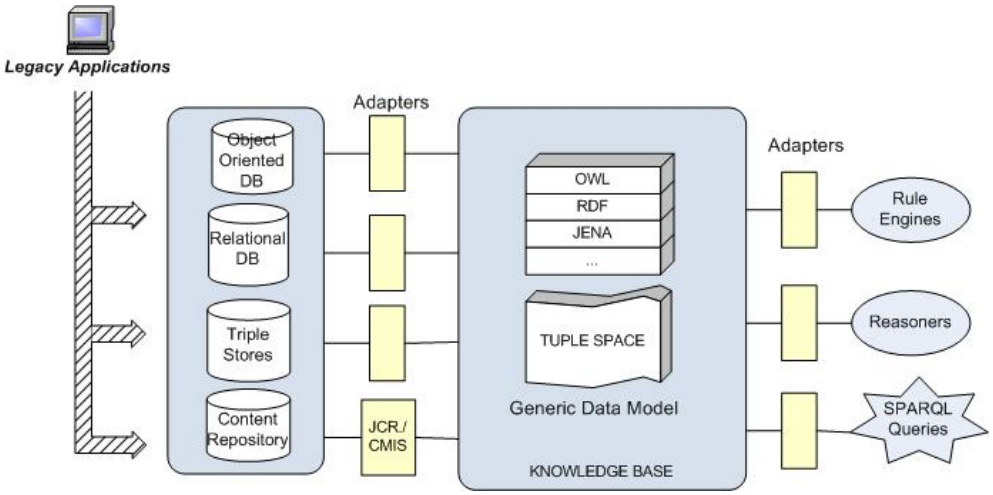



*Figure 2IKS Data Access and Persistence Mechanism*

For this purpose, an engineering approach is taken to develop practical, yet sound persistence architectures which do not exhibit the current, frequently encountered problem of duplication of information in reasoning and database systems. The aim is to provide horizontal pluggable components and APIs (such as semantic indexing mechanisms and semantic search mechanisms) to enable the CMS/KMS providers to develop semantically enabled CMS/KMS systems.

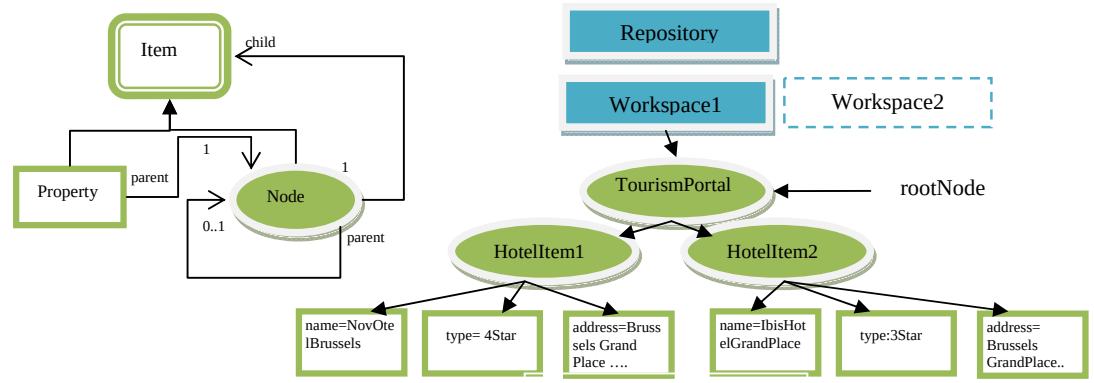
The system components at the data persistency layer of the IKS stack can be summarized as follows:

- Adapters between the traditional persistence mechanisms (relational/object oriented databases, content repositories) and the knowledge bases, enabling semanticizing the already available data in such persistence mechanisms. Communication with the content repositories is over JCR and/or CMIS. The semanticized knowledge can also be enriched with additional rules to better represent the domain knowledge.
- The knowledge base supports plug-in reasoners, rule engines and semantic query languages.
- A triple store to persist the semanticized content in the knowledge base.
- A configurable data synchronization mechanism between knowledge base and traditional persistence mechanisms to feed the inferred explicit knowledge back to these data stores.
- Specific horizontal components and APIs on top of this architecture such as semantic indexing and search of content stored in content repositories.

The first prerequisite of such an architecture is to identify the already available semantics in Content Mechanism, such as content annotation, tagging, indexing mechanisms, and how the metadata is represented and related with content objects. Then automatic synchronization mechanisms should be available to extract this semantics from content repositories, and feed them to a Knowledge base so that semantic search mechanisms can be run on top of this extracted semantics supported through reasoners and rule engines. In this paper we will present such a tool to synchronize the implicit semantics in a JCR Repository with a Persistency Store. We aim to generalize this approach to databases and CMIS supporting content repositories also.

*2.1 IKS JCR to Semantic Persistency Store Synchronization Component*

JCR provides a functional view over the content repository. The content is organized in a tree structure as a hierarchy of "Items", which can be either of type "Node" or "Property" as presented in Figure 3-B. A Property is where the actual data or its associated metadata is stored. On the other hand, a Node, which may have other Nodes or Properties as its children, help application developers build the desired hierarchy over the content. The tree has exactly one root; but the content repository may contain multiple trees called workspaces. Specific Nodes or Properties within a workspace are accessed through XPath expressions. The Properties can either have data type values such as String, Binary or references to other Nodes such as Path, Reference or Name. Through these reference properties the links and relationships between the nodes can also be represented.




A. Information Model B. Example

*Figure 3 JCR Information Model*

In the IKS Semantic Data Access and Persistency Component, we do not aim to replicate the data in JCR Repositories in a Knowledge base, instead, we aim to extract the implicitly expressed semantics in JCR repositories and to store and synchronize them with a knowledge base enabling data persistency mechanisms, reasoning and rule engines. In this way, it becomes possible to use this implicitly defined semantics through horizontal semantic services such as semantic search. For this purpose our first aim is to investigate the already available semantics in JCR repositories.

As can be seen from Figure 3, JCR does not distinguish between "real" content and meta-content, all of them can be represented through the Nodes and Properties. Apart from these constructs, JCR also supports "NodeTypes" which may be used to impose some structural restrictions on content resources. By defining a custom Node Type, one may control the corresponding types of each child Node and/or Property for a particular target Node. The type of a child Node may be restricted to one of the built-in or custom-defined Node Types. On the other hand, for Properties, the user can choose from a predefined set of built-in types only, such as String or Reference. Providing value constraints or default values for a Property is also possible. Finally, an inheritance hierarchy may be defined over custom Node Types where such relationship is expressed by the "supertype" property.

Figure 4 presents a sample declaration of various Node Types in Compact Namespace and Node Type Definition (CND) notation [8]. In this example, a Node of type "iks:HotelDescription" should have a child Property named "iks:facility" whose value must be a String. A value constraint is defined over this Property which restricts the permitted string values to: "Pool", "Disabled Access" and "Sauna". Another Property defined for this Node Type is "iks:sisterHotel" whose value must be a PATH to another Node. A

“iks:HotelDescription” Node Type may optionally contain a child Node of type “cul:ImagesType” which acts as a bag for the associated image URLs. Finally, “iks:HotelDescription” Node Type inherits its structure from the “nt:unstructured” Node Type and enhances it with additional attributes. A formal explanation of the syntax and grammar of CND is provided in [8].

```
<iks = 'http://www.srdc.com.tr/iks'>
[iks:ImagesType]
-iks:imageURL (STRING) *

[iks:HotelDescription]
>nt:unstructured
-iks:facility (STRING) <'Pool', 'Disabled access', 'Sauna'
-iks:sisterHotel (PATH) *
+iks:images [iks:ImagesType]
```

*Figure 4An example Node Type Definition*

A JCR Node Type construct can be considered as an Ontology Class, as it tries to provide a certain schema for the content nodes under the Workspace. From the Node Type definitions and its property and child node declarations, it is possible to construct the Class and the Datatype property and in some cases the Object property definitions. Later on, individuals can be created for the nodes in the Workspace whose primary node type is declared to be these Node Types. The procedure can be summarized as follows:

1. Each JCR node type definition corresponds to an ontology class declaration.
2. JCR properties of type {STRING, LONG, DOUBLE, BOOLEAN, DATE} imply a data property declaration.
3. In ontological formalisms, the JCR properties of type {PATH, REFERENCE, NAME} can be expressed as object properties.
4. It is possible to represent the JCR node type inheritance hierarchy as a set of super class/sub-class relations among ontology classes.
5. When the content repository is populated with instances of the declared types and properties, the generated ontologies can be populated with individuals.

```
<owl:Class rdf:ID="HotelDescription">
   <rdfs:subClassOf rdf:resource="#Resource"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="facility">
   <rdfs:domain rdf:resource="#HotelDescription"/>
<rdfs:range>
   <owl:DataRange>
     <owl:oneOf>
       <rdf:List>
         <rdf:first rdf:datatype="&xsd;String">Pool</rdf:first>
         <rdf:rest>
           <rdf:List>
          <rdf:first rdf:datatype="&xsd;String">Disabled
Access</rdf:first>
         <rdf:rest>
           <rdf:List>
             <rdf:first rdf:datatype="&xsd;String">Sauna</rdf:first>
               <rdf:rest rdf:resource="&rdf;nil" />
                 </rdf:List>
               </rdf:rest>
             </rdf:List>
           </rdf:rest>
          </rdf:List>
     </owl:oneOf>
   </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>
```

```
<owl:ObjectProperty rdf:ID="sisterhotel">
   <rdfs:domain rdf:resource="#HotelDescription"/>
   <rdfs:range rdf:resource ="&owl;Thing"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="images">
   <rdfs:domain rdf:resource="#HotelDescription"/>
   <rdfs:range rdf:resource ="#ImagesType"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="ImagesTyoe">
   <rdfs:subClassOf rdf:resource="#Resource"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="imagesURL">
   <rdfs:domain rdf:resource="#HotelDescription"/>
<rdfs:range rdf:resource ="&xsd;String "/>
</owl:DatatypeProperty>

…..

<HotelDescription rdf:ID="NovotelBrussels">
     <sisterHotel rdf:about="#IbisHotelGrandPlace"/>
     <facility rdf:dataType="&xsd:String">Disabled
Access</facility>
   </HotelDescription>
```

*Figure 5Description of a Node Type and a Node as an Ontology*

In Figure 5, the OWL description of “iks:HotelDescription” Node Type, and an example Node whose primaryNodeType is “iks:HotelDescription” is presented.

Through this methodology, the implicit semantics about the Node Types and the content Nodes can be automatically represented as ontologies. However, another common practice used for annotating the content objects in content repositories is creating a taxonomy hierarchy and relating the content objects with those taxonomy nodes. As presented earlier, JCR does not differentiate content itself from the metadata; i.e. the taxonomy hierarchy can be created as a hierarchy of Nodes with possible Properties. In Figure 6, a taxonomy for annotating tourism services is created as a hierarchy of Nodes, and a content Node defined within a TourismPortal, “NovOtel”, is associated with one of the nodes in this taxonomy, “4StarHotel” through the property “type”.

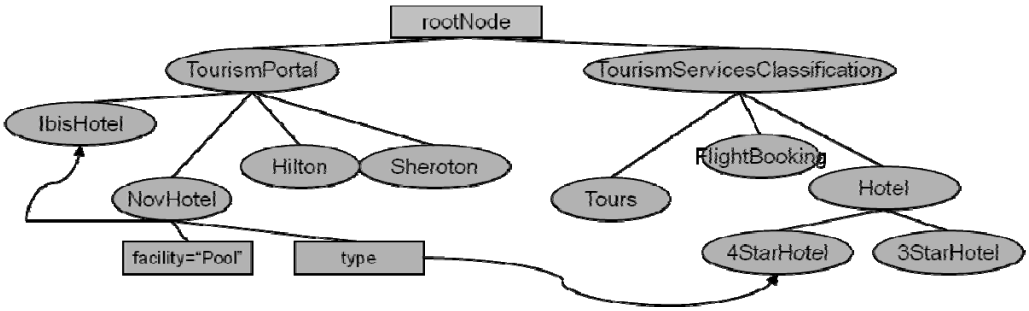



*Figure 6 An example annotation of a Node with a Taxonomy hierarchy*

If we need to represent the implicit semantics of Figure 6 as an ontology, the Nodes in the “TourismServicesClassification” hierarchy should be created as Ontology Classes and “Novotel” content Node should be created as an individual and should be related with an individual of the “4StarHotel” Ontology Class through the “type” objectProperty. In fact, as in our example, if “type” property that links “NovOtel” and “4StarHotel” nodes implies a “ISA” relationship, then it is better to represent the “NovOtel” as an individual of “4StarHotel” ontology class (as presented in Figure 7), which can readily be interpreted by reasoners .

```
<owl:Class rdf:ID=“ToursismServicesClassification”>
 <owl:Class rdf:ID=“Hotel>
   <rdfs:subClassOf
rdf:resource="#TourismServicesClassification"/>
  </owl:Class>
<owl:Class rdf:ID=“4StarHotel>
   <rdfs:subClassOf rdf:resource=“Hotel"/>
  </owl:Class>
……..
```

```
<HoteDescription rdf:ID=“Novotel”>
    <sisterHotel rdf:about=“#IbisHotel”/>
    <facility rdf:dataType=“&xsd:String”>Pool </quality>
   </HotelDescription>
<4StarHotel rdf:about=“#Novotel”/>
```

*Figure 7Ontology representation of the Node Hierarchy presented in Figure 6*

To be able to synchronize this type of semantics defined in a JCR repository with a Persistency Store, a configuration information is required. The Content Repository Administrator should be able to identify the Node Hierarchies within a Workspace that should be represented as Ontology Classes, and the part of the Workspace that should be represented as content individuals. In addition to this, the properties that carry “ISA” relationships should be identified. In IKS Persistency Store Synchronization system, these configuration are handled through a graphical interface. This graphical interface produces a

“Mapping Definition” between the JCR Content Repository to the Persistency Store. Then these mapping definitions are processed by a Mapping Engine to seamlessly lift the implicitly defined semantics in JCR Repositories to Persistency Stores. The Mapping engine also listens to the events in a JCR Repository to keep the JCR Repository and Persistency Store synchronized.

In this way, the semantics of the content objects stored in a JCR Repository can be maintained through a native knowledge base, by storing the metadata as ontology classes, and properties, and creating individuals of these classes for content objects. It should be noted that, the content object themselves are not replicated to the knowledge base, individuals are created for linking them with the ontological metadata and a unique reference to the actual JCR Content Nodes are stored as a datatype property of each content individual. As explained in Section 2, then it becomes possible to enrich this extracted semantics through rule based domain knowledge and to exploit this semantics through horizontal components such as semantic indexing and search by making use of reasoners.

## 3. Business Benefits and Conclusions

The wave of semantic technologies [9] holds significant economic promise for advanced CMS technology providers, but this "wave" has not arrived in Europe, yet, and may never turn into economic success unless we get rid of the current inhibitors. As a part of the IKS Stack, the Data Access and Persistence mechanisms will provide ready-to-use mechanisms to CMS/KMS providers to enable them to semantically enrich their CMS applications and thus, enhance the productivity of CMS end user organisations downstream. In this paper the first component towards this vision is introduced: we have presented how the implicit semantics stored in a JCR enabled Content repository can be semi-automatically extracted and fed in to a knowledge base ready to be exploited by horizontal semantic services such as semantic search.

## References


[1] IST 231527-Interactive Knowledge Stack for small to medium CMS/KMS providers (IKS), http://www.iks-project.eu

[2] “What is Java Content Repository”, http://www.onjava.com/pub/a/onjava/2006/10/04/what-is-java-content-repository.html, Last accessed on February 19, 2009

[3] Content Repository API for Java (JCR) , http://www.day.com/specs/jcr/1.0/, Last accessed on February 19, 2009

[4] OASIS - Proposed Charter for OASIS Content Management Interoperability Services (CMIS) TC , http://xml.coverpages.org/OASIS-CMIS-CharterProposal.html, Last accessed on February 19, 2009

[5] Jena Jena Semantic Web Framework, http://jena.sourceforge.net/, Last accessed on February 19, 2009

[6] Sesame: RDF Schema Querying and Storage, http://www.openrdf.org/, Last accessed on February 19, 2009

[7] *RDFSuite*: High-level Scalable Tools for the Semantic Web, http://139.91.183.30:9090/RDF/, Last accessed on February 19, 2009

[8] Apache Jackrabbit – Node Type Notation. http://jackrabbit.apache.org/node-type-notation.html

[9] Mills Davis, 2008: "Semantic Wave 2008: Industry Roadmap to Web 3.0 and Multibillion Dollar Market Opportunities", http://project10x.com/index.html