Enabling the business-based
Internet of Things and Services

(FP7 257852)

D6.2 State-of-the-Art design and implementation of systems with persistent and high volume data

Published by the ebbits Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme
Objective ICT-2009.1.3: Internet of Things and Enterprise environments

Document version: 1.0 Submission date: 2011-08-31

Document control page

Document file: D6 2_State-of-the-Art_ design_and_implementation_of_systems_with_persistent_and_high_volume_data_v1_0.doc
Document version: 1.0
Document owner: Yves Martin (SAP AG)

Work package: WP6 – Mainstream business systems
Task: T6.1 – Investigation and enhancement of propagation mechanisms for ebbits data
Deliverable type: R

Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Yves Martin (SAP AG)	2011-05-30	Deliverable outline, ToC
0.6	Yves Martin (SAP AG)	2011-07-29	Section 1,2,3,5,6,7 from SAP
0.7	Matts Ahlsén (CNET)	2011-08-19	Section 4 and revision of the other sections. Version for internal review
0.8	Matts Ahlsén (CNET)	2011-08-30	Update after internal review
1.0	Yves Martin (SAP AG)	2011-08-31	Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Karol Furdik (INTERSOFT)	2011-08-26	Approved with some comments
Thomas Kjær (TNM)	2011-08-26	Approved with few comments

Index:

1. Executive summary

This deliverable reports on the state-of-the-art of systems with persistent and high volume data with regards to the ebbits architecture. It describes activities executed in the context of Work Package 6. This work package has as its main task the connection of the various devices / the ebbits middleware to enterprise systems. This report also contributes to the Task T6.1 – "Investigation and enhancement of propagation mechanisms for ebbits data" of the ebbits project. The work in this task will focus on investigation and enhancement of propagation mechanisms for ebbits operational data and its annotations into enterprise systems. This should be done based on research and enhancements on mechanisms that ensure that both primary information as well as additional meta-data (about e.g. data origin, quality, annotations) gets propagated through the overall distributed ebbits setup and can be processed flexibly and with good performance at selectable points within the system.

A section on the requirements on such systems, which follow from the ebbits architecture, constitutes the starting point of this deliverable. The systems should support the data interoperability efforts and the context awareness through semantic annotations within the ebbits project, which is achieved with the help of ontologies used across various application domains and devices and using standard vocabularies in the RDF format. Furthermore, as the ebbits platform should offer an open governance system as an alternative to centralised or centrally owned and administered infrastructure components, the interlinkage of data in these systems should be applicable to a completely distributed system architecture. One cannot assume to be able to provide all necessary links at once; therefore, an incremental approach to link data silos is necessary.

The different types of meta-data which have to be used in the ebbits architecture are described in the Section 4. According to the previously indentified requirements, the RDF representation format is selected and actual technologies for representing data and meta-data in this format and to persist all these data in high volumes in RDF stores are reported in Section 5.

In the next section, a recent approach to interlink data and create added value is presented and aspects for creating RDF data out of relational databases, the automated link generation and the presentation to the user are stated in detail among other technologies.

The technologies, mechanisms, tools and trends identified in this deliverable will form a basis for additional work in Work Package 6. Besides further work in Task 6.1 "Investigation and enhancement of propagation mechanisms for ebbits data", they can support the matching between different information representations in Task 6.2 "Investigation and enhancement of matching services between platform and enterprise systems" and the handling of knowledge enriched data with the meta-data for the enhancement of interaction mechanisms between Business Intelligence functionality in Enterprise Systems and the Distributed Intelligence to be provided by the ebbits platform, which is the content of Task 6.3 "Investigation and enhancement of integration mechanisms for distributed intelligence and business intelligence".

2. Introduction

2.1 Overview of the ebbits project

The ebbits project aims to develop architecture, technologies and processes, which allow businesses to semantically integrate the Internet of Things into mainstream enterprise systems and support interoperable real-world, online end-to-end business applications. It will provide semantic resolution to the Internet of Things and hence present a new bridge between backend enterprise applications, people, services and the physical world, using information generated by tags, sensors, and other devices and performing actions on the real world.

The ebbits platform will support interoperable business applications with context-aware processing of data separated in time and space, information and real-world events (addressing tags, sensors and actuators as services), people and workflows (operator and maintenance crews), optimisation using high-level business rules (energy and cost performance criteria), end-to-end business processes (traceability, lifecycle management), or comprehensive consumer demands (product authentication, trustworthy information, and knowledge sharing).

The ebbits platform will feature a Service-oriented Architecture (SoA) based on open protocols and middleware, effectively transforming every subsystem or device into a web service with semantic resolution. The ebbits platform thus enables the convergence of the Internet of People (IoP), the Internet of Things (IoT) and the Internet of Services (IoS) into the "Internet of People, Things and Services (IoPTS)" for business purposes.

The ebbits platform will be demonstrated in end-to-end business applications featuring connectivity to and online monitoring of a product during its entire lifecycle, i.e. from the early manufacturing stage to its end-of-life. The project will develop, implement and demonstrate two ebbits IoPTS applications. The first application demonstrates real-time optimisation metrics, including energy savings, in manufacturing processes. The other demonstrates online traceability with enhanced information on food.

2.2 Purpose, context and scope of this deliverable

This Deliverable belongs to Work Package 6 of the ebbits project. This work package has as its main task the connection of the various devices / the ebbits middleware to enterprise systems. This includes access to master and operative ERP data, business rules and workflows. Thereby, the information flow should go in both directions, from the enterprise systems to the ebbits middleware and back. Work Package 6 investigates and enhances mechanisms to ensure appropriate integration and interoperability while keeping alternatives for the flexibility and degree (close, loose) of coupling as well as allocation of functionality.

This deliverable reports on the state-of-the art for the design and implementation of systems with persistent and high volume data. It starts with a short section on requirements on such systems which follow from the ebbits architecture. Here mainly two points can be made: The systems should support the data interoperability efforts and the context awareness through semantic annotations within the ebbits project, which is achieved with the help of ontologies used across various application domains and devices. As it was already worked out in the Semantic Knowledge Infrastructure work package (WP4), the W3C standards RDF and OWL should be used for semantic interoperability. Therefore, the focus on this deliverable is also on systems using these standards, so called Triple or RDF stores. The representation and persistence of data and meta-data in RDF and such kind of stores is described in Section 5.

The second point is that the ebbits platform should offer an open governance system as an alternative to centralised or centrally owned and administered infrastructure components. Given additionally the enormous amount of heterogeneous devices, sensors and actuators embedded in systems already existing in the market and the complete distributed architecture of the ebbits project, for the propagation of data within the ebbits setup only an approach with independent

evolution of each single data structure will work. Furthermore, a method with no high up-front cost for integration is desirable. A very recent and more and more popular approach to interlink different data sources with the above properties is the Linked Data methodology. This new approach, which also works very well together with RDF stores, is portrayed in Section 6.

Furthermore, in Section 4 of this deliverable, different forms of meta-data, which will be needed in ebbits project, are described. The Deliverable 3.2 on business vocabularies from Work Package 3 was also taken into account here.

2.3 Background

This deliverable contributes to the Task T6.1 – "Investigation and enhancement of propagation mechanisms for ebbits data" of the ebbits project. The work in this task will focus on investigation and enhancement of propagation mechanisms for ebbits operational data and its annotations into enterprise systems. According to the objectives in the DOW, it is necessary to ensure that information does not get lost but is available in an appropriate and controllable way within the distributed ebbits setup. This is done based on research and enhancements on mechanisms that ensure that both primary information as well as additional meta-data (about e.g. data origin, quality, annotations) gets propagated through the overall distributed ebbits setup and can be processed flexibly and with good performance at selectable points within the system.

3. General Requirements for persistent, high volume Data Systems derived from the ebbits Architecture

In this section, two important features of the architecture of the ebbits project are analyzed with regard to requirements that they entail for systems with persistent and high volume data.

3.1 Semantic Interoperability and Annotation for Context Awareness through Ontologies

According to (Hertel, Broekstra, & Stuckenschmidt, 2008), information access can benefit from the use of ontologies. The employment of ontologies for semantic annotation of device data, data coming from the environment and data from enterprise systems to achieve semantic interoperability is also foreseen in the ebbits project. Providing machine-understandable semantics of data and meta-data enables automatic information processing and exchange. To achieve this, available data and meta-data in the ebbits project have to be linked to concepts and relations in the corresponding ontologies, which are mainly worked on and provided by Work Package 4. In the deliverables published in this work package so far (D4.1, D4.2 and D4.3), the W3C standards RDF and to a lesser degree also OWL are recommended for the creation of ontologies.

Additionally to conception of ontologies, access mechanisms have to be provided that support the integrated model consisting of ontology and data. The most common approach to combine data with ontologies is via an RDF representation of available data and meta-data that describes the data as instances of the corresponding ontologies and also the meta-data in the same RDF format. Once all this information is represented using this uniform data structure, it will be easier to interchange the data between different machines, devices and systems. To accomplish this, also conventional data from relational databases have to be mapped to the RDF format. Fortunately, there already exist semi-automatic approaches for this task (see also Section 6.2 below).

Today's technologies for the persistent RDF data storage are well developed and able to handle really huge amount of data (for an example see Section 5.4 below). The general technologies to represent data and meta-data in RDF format and to persist it in RDF stores are given in Section 5.

3.2 Distributed Setup and the Propagation of Data

According to the Description of Work of the ebbits project, the to-be-developed ebbits architecture is inherently distributed; there will be data and meta-data from various sources and different devices, sensors, machines and enterprise systems. Therefore, the ebbits architecture should be an open system as an alternative to centralized or centrally owned and administered infrastructure components. The ebbits platform may thus eliminate centralized gatekeeper lock-in of critical business or process functionalities. The propagation of data and the distributed processing at different points within the distributed system should be achieved by interlinking the various data silos. An approach to realize these links, which as well operates on the chosen RDF format, is called Linked Data (Berners-Lee, 2006). This methodology is described in Section 6 including mapping relation data from databases into the RDF format, publishing/creating links, validating web servers, navigating the linked data and displaying it to the user. This approach has some advantages with regard to the ebbits architecture. It allows for the independent publication of links and furthermore, the data structures can involve independently from each other. There is no need for a central control system. Linked Data uses the so-called pay-as-you-go data integration method. That is, there is no need to have a complete solution from the beginning, one can start small and add more and more links as it is needed. The data is machine-readable, so it is possible to build automated services on top of it. The information is linked to instead of copied which provides for the most up-to-date data, prevents inconsistencies and saves data management costs.

The Linked Data approach promises to create an interoperable layer on top of emerging, distributed ebbits architecture. Vocabularies and ontologies are used to represent resources and data according to Linked Data standards and in the RDF format. Unified query processing, propagation and link-following access methods are given for highly dynamic data across a multitude of sources.

4. Meta-data in ebbits

With a focus on the integration of business systems with the IoPTS, the ebbits platform will need to exploit many different types of meta-data, in terms of both different properties as well their representations (formats).

4.1 Data and Meta-data

A fundamental aspect of any data set or source is its own description. The (meta) data used for such descriptions can be based on vocabularies specifically designed to convey the meaning from a certain perspective, or an intended use of the data in general terms or for a specific application domain. Most organisations and companies have a long tradition of meta-data management, e.g., related to enterprise business models and the design of databases as well as internal and external web sites. Meta data management is an intrinsic property of any ICT systems design and operation today.

Methods and technologies for meta-data are used to improve search capabilities, as a complement to free text indexing of contents and advanced results ranking mechanisms. Semantic descriptions of the components of IT-systems have always been an important issue, although often neglected. For the past fifteen years the growth of the Internet and the emergence of standards for content management like XML and related technologies (W3C 2011) have made the issue of semantic interoperability a global one (EC 2010). This in itself requires proper modelling and use of meta-data in any application or service.

Specific domains, like the library and the publishing sectors, also have a long history in managing meta-data but from another perspective, which primarily has focused the identification and copy right protection of immaterial works. One concrete result of this was the Dublin Core[1] vocabulary for describing digital documents on the Internet (see below), an early attempt to provide a global meta-data facility for the Internet.

Thus, we use meta-data to convey an intended meaning of data. The interpretation is however always dependent on context, the same data item could be mapped to multiple meta-data descriptions (classified in multiple taxonomies) depending on context and the type of application.

4.2 Modelling and Representing Meta-data.

The target of a meta-data description is commonly referred to as *resources,* where a resource could be any abstract or concrete phenomenon, represented directly or indirectly in digital form. In ebbits, resources may correspond to various types of devices and their services as well as product databases and logs. Users, being actors in various processes can also be considered as resources amendable to meta-data descriptions. The resource concept is used in the RDF, the Resource Description Framework from W3C (described in detail in the Section 5.1), as a central object in the RDF data model.

In ebbits the semantic models used in meta-data description will primarily be based on ontologies. Ontologies are a means to represent vocabularies.

There are many existing vocabularies (with varying degrees of specialization) in many different industry sectors and application domains (deliverable D3.2 provides a comprehensive overview of several such vocabularies with relevance to ebbits). A well-known example intended for meta-data annotation of web resources is the Dublin Core (DC) initiative, which has roots in the publishing sector. The Dublin Core (DC) Metadata Element Set is a vocabulary consisting of fifteen properties (or elements) intended for descriptions of primarily digital documents, where the elements can describe creators, dates, formats etc. As of 2009 the DC element set is endorsed as an ISO standard

[1] http://dublincore.org/

(ISO 2009). The fifteen elements Dublin Core[2] set described in this standard is part of a larger set of metadata vocabularies and technical specifications maintained by the Dublin Core Metadata Initiative (DCMI). The full set of vocabularies, DCMI Metadata Terms also includes sets of resource classes including a type system and encoding schemes. The terms in DCMI vocabularies are intended to be used in combination with terms from other vocabularies[3] and standards[4], and in different meta-data representation frameworks, such as the RDF.

When adopting solutions based on ontologies and vocabularies, issues of sharing and control need to be considered. Both ontologies and vocabularies are by definition shared specifications, used by different stakeholders. The responsibilities and procedures for model evolution and maintenance should be clarified. The term *control* in controlled vocabularies, here implies that there is some authority which maintains the vocabulary/ontology and provides guidelines for use (in the case of the DC vocabulary, the DCMI is such an authority). This is an important aspect, e.g., with respect to ensuring data quality (see below). The use of formal standards to this end is an advantage.

The choice of model and representation formalism obviously affects the possibilities of subsequent processing and interpretation of data and its meta-data. Adopting technology for ontology use brings with it the possibility of exploiting reasoning capabilities, i.e., the possibility to derive "new" facts (data) by means of inferencing. In most cases this basically means the traversal of class taxonomies. More sophisticated reasoning requires more elaborate formalisms and processing power, which adds to the complexity of the models.

4.3 Categories of Meta-data in ebbits

A starting point for an ebbits meta-data architecture is a set of basic categories of meta-data. On a highest level we can of course distinguish between generic and domain (application) specific meta-data, which can be further specialized into subsets. However, there is not always a distinct separation between what is generic and domain specific. A further elaboration of a meta-data architecture should also map the different categories to the different abstraction layers of the overall ebbits platform architecture.

4.3.1 Generic Meta-data

In principle we can distinguish at least the following generic categories with meta-data for,

- Device classification
- Events and services
- Network status

The ebbits platform integrates physical devices thru its so-called Physical Adaptation Layer (PWAL) (Alcaraz et al. 2011). To support this integration, devices are classified according to different properties such as their processing capabilities. For this purpose, an ebbits Device Ontology provides the necessary meta-data concepts and terms for classification of all ebbits devices. This ontology is based on a corresponding ontology adapted from the Hydra/LinkSmart middleware.

[2] The name "Dublin" originates from a 1995 workshop in Dublin, Ohio, and "core" because its elements are generic, usable for describing a wide range of resources.
[3] E.g., subject headings from a library/publisher
[4] Like various international standards, e.g., representing date and time

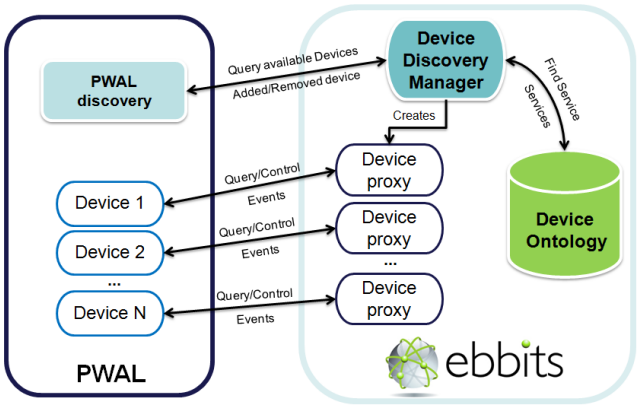


Figure 1: Meta data in the device ontology is used for the discovery and integration of physical devices into the ebbits platform

A device ontology has several different usages, and can, e.g., be used to support application development with ebbits devices.

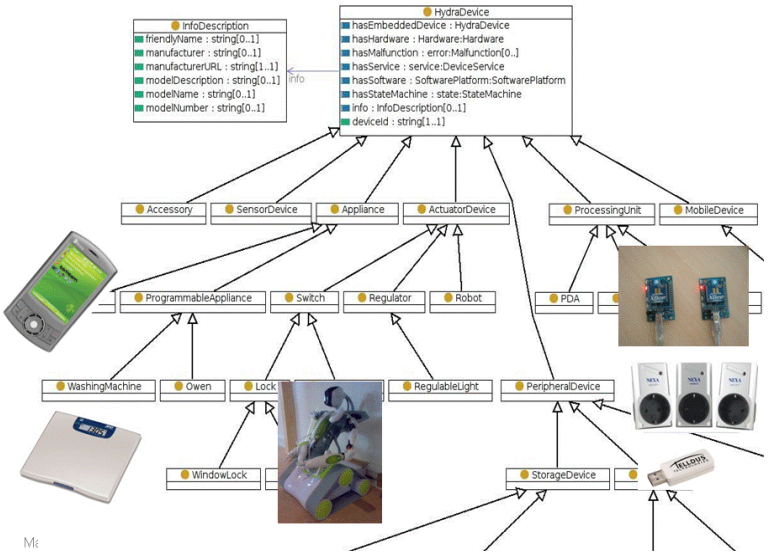


Figure 2: Subset of a device taxonomy, from Hydra (Ahlsen et al. 2010).

The Event Core Ontology encodes the meta-data which will be used to annotate different types of events and services in the ebbits architecture. The first version of this model is specified in deliverable D7.2 (Kostelnik et al. 2011), and describes the meta data in terms of the concepts and terms, which can be used to classify and relate sensors, events and services in the ebbits architecture. This includes a core taxonomy of events, models of event results, and event stimuli (e.g. the occurrence of the real-world situations triggering events), event capabilities, as well as knowledge about the data storage used to store the event results.

Meta-data related to the network status, mainly refers to properties like topology, link bandwidth, type of physical carriers (wired/wireless), reliability and transmission security. Additional meta-data may also be used to describe whether and in what way the net supports the opportunistic networking[5] (Kool et al. 2011).

[5] Opportunistic networking may include support for: delay tolerance, frequency agility and multiple wireless technologies.

4.3.2 Domain-specific Meta-data

The domain-specific meta-data will be drawn from the initially selected application scenarios (or domains initially elaborated in deliverable D3.1) (Checcozzo et al. 2011): the manufacturing domain with a view towards energy efficiency applications, and the agriculture domain, where the focus is on traceability issues along the food production chain.

During the projects first iteration of requirements elicitation, two main standard frameworks relating to the two domain have been analysed within WP6 (and WP7), for manufacturing the ISA-95 standard for ERP-MES integration (IEC/ISO 2004) (IEC/ISO 2003), and for the agriculture domain, the ISO agriculture data element directory, ADED, and the related ISO standard for data interchange ADIS (Kostelnik et al. 2011).

The selection of relevant subsets of meta-data properties for the two domains is subject to further analysis, and specifically to be coordinated with elaboration of business vocabularies in WP3. The deliverable D3.2 (Sabol et al. 2011) provides a compilation of several vocabularies and standards from the two scenarios (in addition to the above mentioned), and also provides an analysis of their applicability cross domains.

It is expected that some of the meta-data properties from these domain standards will be built into the generic part of the ebbits platform meta-data architecture as the platform evolves.

4.3.3 Data Origin and Data Quality

Considering the application of ebbits with respect to traceability, two aspects of meta-data are particularly important to capture: (1) the source or origins of data and (2) the trace route of data/messages thru a deployed ebbits platform. To support this, a meta-data model for ebbits should include defined concepts for entities representing data originators (sources, creators, producers etc.) and data consumers or clients (e.g., including end users, devices, applications). This issue is related to the provision of functions for authentication and for the identification of objects (resources) in the ebbits platform. These are architectural issues currently under investigation, and are not further elaborated in this document.

The ability to securely verify the source and origins of data is also one prerequisite for determining data quality.

Data quality can in general terms be understood as how well data meets an intended use in a specified context or for a specific purpose (or its "fitness for use") (Gustafsson et al. 2004).

There is both a subjective and an objective dimension to data quality. Subjective in the sense that quality is at the judgement of the stakeholders (user, organisation etc.), and may thus vary in between contexts and stakeholders. Quality properties/attributes and quality measures and procedure, must however be objective.

From the ebbits perspective, we should consider quality with respect to both the specifications of data/services, as well the quality properties of the data/services.

Quality of specifications mainly refers to,

- The definitions of data, in terms of coverage and expressive power.
- The availability/accessibility of the specifications and directories.
- Use of and adherence to applicable standards.
- The existence of responsible authorities for the specifications (c.f., controlled vocabularies).

The quality properties of data/services refer to,

- Completeness, i.e., the extent to which data models a domain or phenomenon, totally or partially.
- Accuracy or reliability, e.g., weather data correctly represents the event or object intended.
- Precision or granularity, i.e., the exactness or level of detail

- Timeliness, i.e., how current data is.
- Trust or credibility, i.e., trustworthiness. This may be based on third party statements regarding a resource e.g., expressed in RDF.

For ebbits, the meta-data models used, encoded in one or more ontologies, will be the basis for ensuring data quality. The current choice of modelling and representation framework is the RDF in combination with OWL.

The choice of the specific metrics and value sets for the above quality properties is subject to further investigation, and should be kept open for domain specific adaptations or specializations.

4.4 Annotations

By *annotations* we mean the mechanisms to associate meta-data with its target, i.e., various resources in terms of data, events and services.

In principle, annotations are either embedded with a resource or represented by a reference from the resource to a semantic description, the latter expressed in an ontology or in some subset thereof. For example, the Meta tags, or DC elements in an HTML page make up an embedded meta-data annotation, e.g., describing authorship, creation dates. RDF descriptions (detailed in the next section) could be either embedded or given by reference, e.g., to a class taxonomy in a separate ontology.

In the case where a meta-data description is maintained independently of both the resource in question and the semantic model used by the description, we do not speak of an annotation, but of independent resource descriptions (this is was the initial intended use of the RDF approach).

The ebbits platform should support the annotation of data, events as well as services. Data objects will primarily be annotated using the RDF formalism in combination with various vocabularies.

The ebbits architecture is very much focused around data fusion and event management, where semantic annotation with meta-data plays an important role. The architecture style of ebbits can be characterized as *Event-driven SOA*, integrating intelligent services with advanced semantic event processing and business rules. Thus events can range from lower level atomic signals to higher level semantically enriched message carriers. On a higher abstraction layer ebbits events are mapped to business rules, which can make use of intelligent services, to implement the business logic for reporting, actuation of devices, as well as for further event generation.

In the ebbits system, events triggered by e.g., a sensor, will be propagated thru several system levels and can be stepwise refined (or semantically enriched) by means of meta data annotations based on an ebbits event core ontology (Kostelnik et al. 2011).

Among the motivations for annotation of services are to facilitate service discovery. This can e.g., be used to find a device with some specific service capabilities, perhaps as a replacement for faulty one (where the fault was captured by an event being caught).

SAWSDL, Semantic Annotations for WSDL and XML Schema (W3C 2007), is an example of an annotation mechanism for associating semantic descriptions to web services. It is an extension to the well-known WSDL (Web Service Description Language) format for web service definition. SAWSDL is a W3C recommendation that defines how to add semantic annotations to WSDL files and to XML Schema elements. It defines the extension attributes that can be applied to elements in both WSDL and XML Schema in order to annotate WSDL interfaces, operations and their input and output messages. SAWSDL is the successor of WSDL-S often considered the first step towards (de facto) standardization in the area of Semantic Web Services. Semantic annotations in WSDL and XML Schema are used for these purposes:

- associating web service interfaces with semantic models (ontologies and vocabularies) to facilitate semantic Web service discovery,
- describing the purpose or applicability of web service operations to ease discovery or composition,

- linking and mapping inputs, outputs and faults of web service operations to semantic concepts to help facilitate mediation and service discovery and composition.

According to the W3C SAWSDL Working Group (http://www.w3.org/2002/ws/sawsdl/), the key design principles for SAWSDL were:

- The specification enables semantic annotations for Web services using and building on the existing extensibility framework of WSDL.
- It is agnostic to semantic representation languages (i.e., meta-data model and formalism).
- It enables semantic annotations for Web services not only for discovering Web services but also for invoking them.

SAWSDL can be split in two extension components: (a) An extension attribute, named *modelReference*, to specify the association between a WSDL component and a concept in some semantic model (such as an ebbits ontology) and (b) two extension attributes, named *liftingSchemaMapping* and *loweringSchemaMapping*, which are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML.

The Model reference is the first major part of SAWSDL represented by the attribute called *modelReference*. The value of the attribute is a list of URIs that reference concepts in a (possibly external) semantic model (such an ebbits device ontology). SAWSDL defines how model references can be used on WSDL interfaces, operations, faults, and on XML Schema element declarations or type definitions.

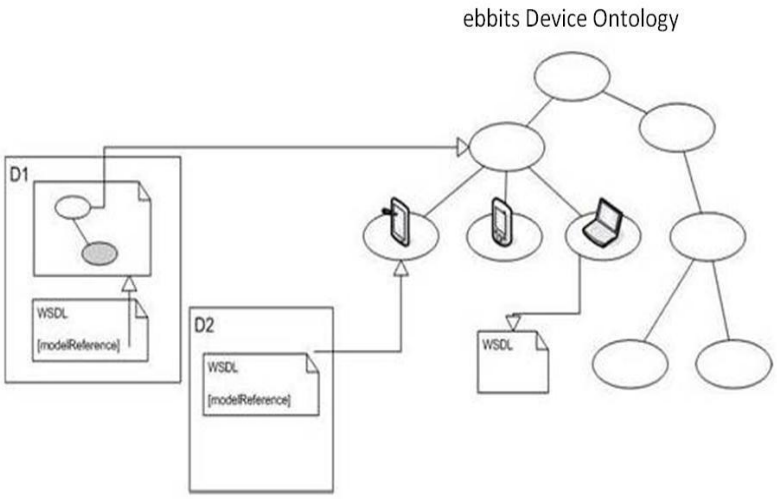


Figure 3: Devices with service descriptions (wsdl) annotated with meta data reference to an ontology.

On a WSDL interface, a model reference can provide a classification of the interface, for example by pointing into a products-and-services taxonomy. Model references on a WSDL operation define what the operation does. This can be done with a direct reference to a verb concept or to a logical axiom or by specifying the operation's preconditions and effects. On a WSDL fault, model references define the kind of failure the fault means, so that the fault can be handled more appropriately by the client. Model references on XML Schema element declarations and type definitions define the semantics of the inputs or outputs of WSDL operations. In general, model references can have many uses, and indeed, SAWSDL does not limit the applicability of the attribute.

The second part deals with Schema mappings that transform between XML data described with XML Schema, and, descriptions in a semantic model. The mappings can be used for example to support invocation of a Web service from a client that works natively with semantic data. SAWSDL defines two extension attributes: *liftingSchemaMapping* and *lowerSchemaMapping*. These attributes are used to point from a schema element declaration or type definition to a mapping that specifies (in any suitable mapping language, e.g. XSLT) how data is transformed from XML to the semantic level (*lifting*) or back (*lowering*).

5. Representation Format for Data and Meta-data

As it was pointed out in Section 3, the RDF format seems to be suited for storing and propagating data and meta-data with the ebbits project. In this section, a short introduction to the format and surrounding concepts is given. Additionally, it is shown how to represent data and meta-data in this formal language. Furthermore, it is described how persistence of high volumes of RDF data is achieved and an RDF store is presented. Some content of this section was already published in Deliverable D4.1 "Analysis of Semantic Stores and Specific ebbits Use Cases" (Knechtel et al, 2011) and Deliverable D4.3 "Coverage and scope definition of a semantic knowledge model" (Furdik et al, 2011).

5.1 Resources, URIs and the RDF Format

The RDF (Resource Description Framework) is a main component of the semantic web and is a family of standards of the World Wide Web Consortiums (W3C) for the formal description of information about objects, so-called resources. Hereby, a resource can be abstract or real. Resources in the web are, for example, documents, pictures, files, services, email or other things (W3C, 2011). A resource does not have to be nessecarily reachable in the internet, for example also humans, companies or hardcover books are resources. Mathematical equations, numbers or properties like "being a parent" are also resources but abstract ones (Berners-Lee, 2005). The conceptual mapping to the entity is dependent for abstract resources on the time and other factors. For example, a resource may represent a company logo. When a request for the representation of that logo is issued, the particular representation the request receives may depend on a number of factors such as time (the logo may change over time) and the file format required (McBride, 2001).

Resources are adressed by unique identifiers in form of so-called Uniform Resource Identifiers (URI). URIs are strings, which follow a certain syntax and are used as universal standard for adressing abstract and real resources. Syntactily, URIs consist of several parts, thereby the first part is seperated with a colon from the remaining and denotes the type of an URI (Hitzler et al, 2008). For this so-called schema there are a lot of examples in the web like http, ftp or mailto. The URI schema serves for the interpretation of the remaining part, like for example in mailto:Joe@example.com, where the second part is indentified as an email address.

Furthermore, URIs can be classified as Uniform Resource Names (URN) or as Uniform Resource Locators (URL). URNs are used for unique naming and are not retrievable online, while URLs give the location of the resource and can be retrieved by using the corresponding protocol.

The RDF is a formal language for the description of structured information. Through this framework, applications should be enabled to exchange data on the internet without having the data losing its original meaning (Hitzler et al, 2008). A RDF document is a collection of RDF-statements, so-called triples. Such a RDF-triple consists of a subject, a predicate and an object. A set of triples is called a RDF-graph. A triple denotes a relationship between the subject and the object, where the relationship is directed from the former to the latter and is labelled by the predicate, which is a binary relation. An RDF-graph is a directed graph, that is, a set of nodes which are connected by directed arcs.

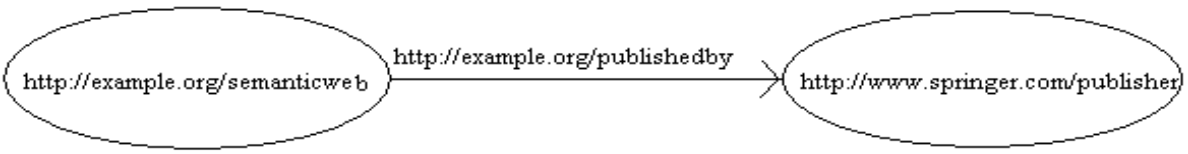


Figure 4: A very simple RDF graph

In Figure 4, a simple RDF graph is depicted that denotes the relationship between a book over the semantic web and the publisher Springer. The predicate expresses that this book was published by Springer.

The subject and the predicate in a RDF-triple are always resources and are denoted by URIs. The object in a triple can be a resource or a so-called literal. Literals denote data values in RDF and they are reserved identifiers for RDF-resources of a specific data type (Hitzler et al, 2008). Literals are strings, which are sometimes interpreted on the basis of a given data type. In this way, it is possible to specify truth values (true or false), numbers (for example, 42 or 0x2A) or dates (for example, 2009-02-01-9:00).

A peculiarity are so-called blank nodes, which do not have a URI and therefore, are not linked to the originally described resource. These nodes are used as supporting nodes, which point to structural resources (Hitzler et al, 2008). Such a construction is necessary for the description of more than binary relations. For example, in Figure 5 such a blank node is used to construct a unique multivalent relation between a staff member and its address with city, street, state and postal code. Such structural resources can also be seen as existentially qualified variables (McBride, 2001).

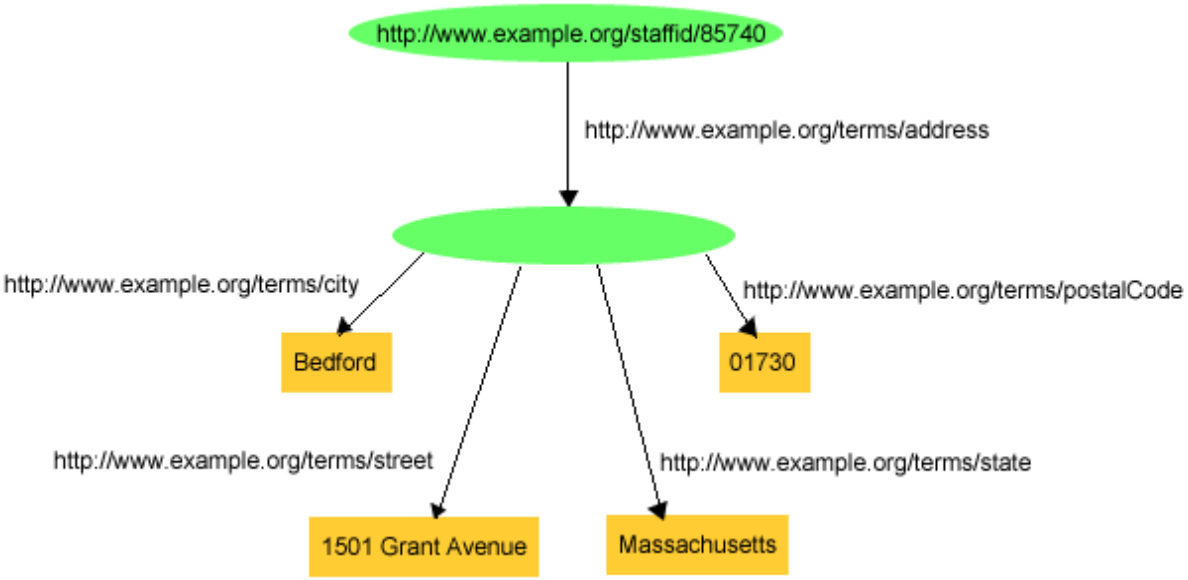


Figure 5: Usage of blank nodes in RDF, Source (W3C, 2011)

5.2 Data in RDF

Two examples are given, one for the manufacturing domain and one for the food traceability domain on how to represent different data for a piece in the production and for a pig farm in RDF:

@prefix : <http://www.example.org/sample.rdfs#> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<!-- Data for production scenario -->

:Piece rdf:type rdfs:Class;

rdfs:subClassOf :Product .

:Device rdf:type rdfs:Class;

```
rdfs:subClassOf :Machine.

:hasStatus rdf:type rdf:Property;
rdfs:domain :Piece;
rdfs:range :xsd:string .

:hasLog rdf:type :ProductionStatus;
rdfs:domain :Piece;
rdfs:range : ProductionStatus .

:log1 rdf:type : ProductionStatus;
:hasTimestamp "2011/06/01 13:24:56"^^xsd:datetime ;
:hasStatus "in-progress"^^xsd:string ;
<!--                                    Data for food scenario          -->
:Pig rdf:type rdfs:Class;
rdfs:subClassOf :mammal .
:hasId rdf:type rdf:Property;
rdfs:domain :Pig;
rdfs:range xsd:integer .
:hasMother rdf:type rdf:Property;
rdfs:domain :Pig;
rdfs:range :Pig .
:hasWeight rdf:type rdf:Property;
rdfs:domain :Pig;
rdfs:range xsd:integer .

:pig1 rdf:type :Pig ;
:hasId "1"^^xsd:integer ;
:hasMother :pig10;
:hasWeight "5"^^xsd:integer .
```

5.3 Metadata in RDF

The metadata, which is used in the ebbits project and which is described in Section 4, has also to be encoded in RDF. A first possibility to do this uses a special means of expression, the so-called reification of triples. Normally, RDF statements itself are not resources. Through reification, there are resources that represent RDF statements, that is, a statement is assigned a URI and treated as a resource about which additional statements can be made (McBride, 2001). The technique used is similar to the above described method to represent multivalent relations (Hitzler et al, 2008). For the triple, about which additional statements should be made, a structural resource is introduced. If this new resource should only be used locally, a blank node would also be sufficient. In any case, the

resource is assigned the type `rdf:statement`. Additionally, this structural resource is the subject of three additional triples, which relates to it the subject, predicate and object of the original triple. The corresponding predicates are, `rdf:subject`, `rdf:predicate` and `rdf:object`, respectively. Now this structural resource can be the object in other RDF-triples and in this way, additional statements can be made about the reified statement, as for example in "Alice says that Bob has the secure key K". In such a way, reification can be used to specify the origin of the data or to denote the data quality.

The second possibility are so-called named graphs, in which a set of triples is named by a URI, or using RDF quads instead of RDF triples. For RDF quads each triple is augmented to a quad by also storing a context value, which can be the name of the graph where the triple originated from. There are small differences between RDF quads and named graphs (Bizer & Carroll, 2004). In named graphs, the graph is not treated with the open world assumption, that is, the assertion of a named graph asserts that this graph contains exactly the triples given, and there are not any others triples that have been omitted. A context in a quad is subject resource, which may have additional properties not mentioned in a document.

Both, RDF quads and named graphs, can be used to represent context without the need to reify the triples. Besides recording the data origin in the context value of a quad or as name of the graph, it would also be possible to do trust evaluations and compute a measure for the quality of the data. For this approach, it would be necessary to have provenance meta-information about the context in which a triple has been published, e.g. who said what, when and why (Bizer & Carroll, 2004). Meta-information about the author of the named graph could be used for role-based trust policies like "Distrust everything a vendor says about its competitor." As a simple example for named graphs in the ebbits production scenario, consider the following,

```
@prefix : <http://www.example.org/sample.rdfs#> .
@prefix dc: <http://purl.org/dc/elements/1.1/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

G1  (:log1 rdf:type : ProductionStatus;
      :hasStatus "in-progress"^^xsd:string ;
      :hasDevice :Device )

G2  (:log1 rdf:type : ProductionStatus;
       :hasStatus "finished"^^xsd:string ;
       :hasDevice :Device )

G3  (G1 dc:author  :device1;
       G1 dc:date "06/06/2011";
       G2 dc:author  :device2;
       G2 dc:date "07/06/2011")
```

where there are 3 named graphs. This first graph stems from one device and states that the production is in progress, the second one from a different device asserts that production is finished. The third graph gives some meta-data about the first two ones and if one prefers newer information about older data, the conclusion would be to trust the data from the second device.

5.4 Persistence of RDF Data in Stores

Current state-of-the-art storage and retrieval technologies of RDF data use so-called RDF stores or Triple stores, which allow storage of such data and provide methods to access that information (Hertel, Broekstra, & Stuckenschmidt, 2008). Such an RDF store consists of two main components, a

repository and a middleware on top of this repository. The general layered architecture is shown in Figure 6.

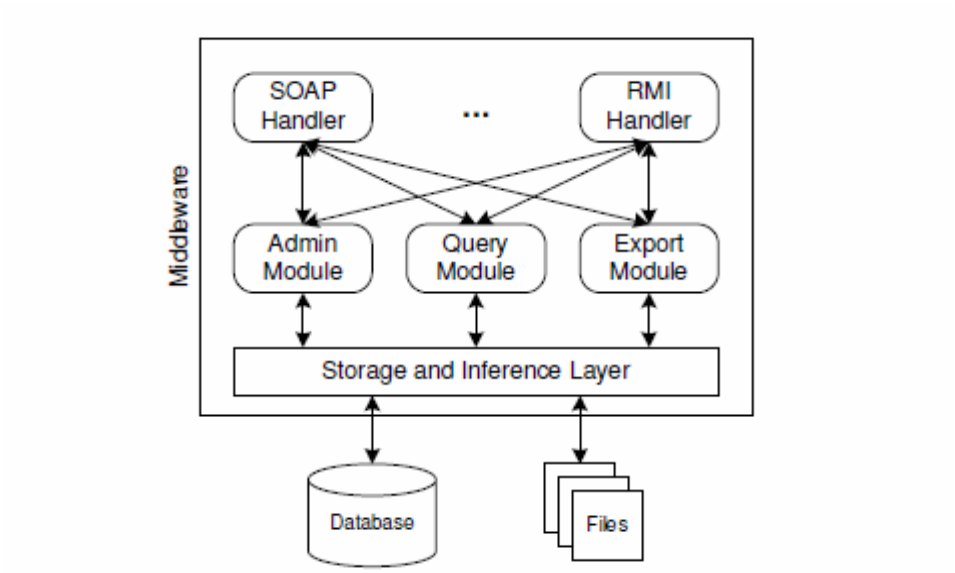


Figure 6: Generic Architecture of an RDF Store, Source: (Hertel, Broekstra, & Stuckenschmidt, 2008)

The bottom layer, the Storage and Inference Layer (SAIL) encapsulates the access to the repository, be it main memory, files or a database. Furthermore, this layer also provides for some inferencing, for example, to compute the transitive closure of the class hierarchy. One important aspect here is the time, when the inference should be executed (Hertel, Broekstra, & Stuckenschmidt, 2008). It is possible to do this in advance, the so-called eager evaluation, or at the query time, which is denoted as lazy evaluation. Eager evaluation reduces the time needed to answer queries but can greatly increase the number of RDF statements which have to be stored. Furthermore, some of these statements may never be in the resulting answer set of any query. This is avoided by the lazy evaluation which only generates used entailments but as a disadvantage the query processing time can be increased significantly. A combination is possible where the entailments with fewer consequences are computed beforehand and entailments with less evaluation time are executed on-the-fly.

The layer above depicts the middleware which is responsible for providing functionality for adding, deleting, querying and exporting data. It consists of three modules (Hertel, Broekstra, & Stuckenschmidt, 2008):

Admin Module: Provides methods to add or delete data from the RDF store. New data can be created in main memory and by using a function be stored into the knowledge base or loaded from files. Especially loading data usually also involves parsing and validating the new triples. Therefore, a RDF parser and a RDF validator is part of this module. Most parsers allow for the RDF/XML notation. The validator checks the incoming data for correctness and also for compliance with already loaded ontologies. Delete operations are easy to implement if the complete store is erased. If only single RDF statements should be removed, this could entail the removal of other triples which have been inferred in the transitive closure of the corresponding statement. In this case, the deletion operation can become costly.

Query Module: It is responsible for handling queries to the RDF store. If more than one query language should be supported, different query modules with parsers and handlers may be necessary. Most of the RDF stores support the SPARQL query language which is an official W3C Recommendation. The query syntax is analyzed by the parser and possibly translated into a different model to capture the query semantics. Afterwards, the SQL query is formed and sent to the database. The syntax of the resulting SQL query is often dependent on the underlying database. That is one of the reasons for the intermediate SAIL layer which abstracts from such implementation details and provides for a uniform access to the storage mechanisms. Most RDF stores leave the task

of query optimization to the database system as modern systems have sophisticated evaluation and optimization mechanisms. In such a case, the query must be translated to SQL as completely as possible.

Export Module: If one wants to exchange the data with other systems, this module allows for a dump of the RDF data into a file. Thereby, the data can be serialized into different notation.

On the top, there is a layer with modules for the local/remote access to the middleware, for example for SOAP calls or RMI.

Within a repository of an RDF store it would only be possible for small amounts of data to serialize them to files, usually a database management system is used. In general it is possible to use relational databases or object database management systems. In practice, relational database management systems (RDBMS) are predominant (Hertel, Broekstra, & Stuckenschmidt, 2008). Storing RDF data into such a RDBMS requires an appropriate table design. One can distinguish two general classes of approaches, one is working with generic schemas, which do not depend on the ontology and the other takes ontology specific schemas. These different approaches are further described in the following (Hertel, Broekstra, & Stuckenschmidt, 2008):

Generic Schemas: The simplest generic schema uses just one table, called Triples, with three columns for subject, predicate and objects and therefore mirrors the triple nature of RDF statements. It has the advantage that no restructuring is needed if the ontology changes. On the other hand, performing a query requires a search through the whole database which can become very costly, especially when joins have to be performed. A first improvement is the separate storage of resource URIs and literals in two new tables and substituting them with IDs in the original table. This reduces the amount of storage needed for all the data. A further refinement is the horizontal splitting of the Triples table into several tables, one for each RDF property. The new tables only contain two columns for subject and object and the predicate is implicit from the table name. The inference of all instances of a class, for example, can now be computed by a look-up in just one smaller table.

Ontology Specific Schemas: When the ontology is changing (classes or properties are added/deleted), these schemas have to be adjusted too. There are two main ontology specific schemas. One is called the one-table-per-class schema, where a table for each class is created which has columns for all properties whose domain contains this class. Another one is called one-table-per-property schema, where there is one table per property. In practice, these two schemas are often combined in a hybrid schema, where the tables from schemas are used and the resources are substituted by unique IDs. The hybrid approach has the advantage that adding new classes and properties do not lead to changes in existing tables; i.e., instead of changing potentially a lot of existing tables rather a new table is created.

As an example for an RDF store, which can persist very high volumes of RDF data, the BigOWLIM is briefly introduced in the following (for details see the description in D4.1 (Knechtel et al, 2011)):

BigOWLIM[6] is a high-performance semantic repository, implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database that scales to massive quantities of data. BigOWLIM can manage billions of statements and multiple simultaneous user sessions. The key features of BigOWLIM are as follows:

- The most scalable semantic repository in the World, both in terms of the volume of RDF data it can store and the speed with which it can load and do inferencing;
- Pure Java implementation, ensuring ease of deployment and portability;
- Compatible with Sesame 2, which brings interoperability benefits and support for all major RDF syntaxes and query languages;
- Customisable reasoning
- Optimized owl:sameAs handling, which delivers dramatic improvements in performance and usability when huge volumes of data from multiple sources are integrated.

[6] http://www.ontotext.com/owlim/

- Clustering support brings resilience, failover and scalable parallel query processing;
- Geo-spatial extensions;
- Full-text search support;
- High performance retraction of statements and their inferences – so inference materialisation speeds up retrieval, but without delete performance degradation;
- Powerful and expressive consistency/integrity constraint checking mechanisms;
- RDF Priming, based upon activation spreading, allows efficient data selection and context-aware query answering for handling huge datasets;
- Notification mechanism, to allow clients to react to statements in the update stream.

6. Propagation of Data and the Linked Data Approach

Data and Meta-data in machine-readable form should be propagated in the very distributed ebbits architecture. As already mentioned in Section 3, a new and promising state-of-the-art approach to achieve this is Linked Data. In this section, the basic principles and advantages of Linked Data are described. Additionally, some preconditions for this approach are also given. Furthermore, this section also illustrates how to transform data in a database into RDF and also how to generate links between these new RDF resources. How to validate the published Linked data is also explained in this section. The section concludes with a description of ways to browse the interlinked information and to present these now machine-readable data also to humans.

6.1 Principles and Assessment

The four Linked Data design principles have been initially proposed by Tim Berners-Lee (Berners-Lee, 2006):

1. **Use URIs as names for things:** for example, http://www.sap.com/employees/yves_martin
2. **Use HTTP URIs so that people can look up those names:** For example, one can retrieve data about the person above via an HTTP-GET.
3. **When someone looks up a URI, provide useful information, using standards:** For example, the description about the person is in RDF and can be accessed using the SPARQL language.
4. **Include links to other URIs, so that they can discover more things**: The description contains URIs of the lab the person is working in, the projects, the manager etc.

The principles are seen as best practices to publish and interlink data in a way that maximizes reuse and in doing so, add additional value to them. If one uses Linked Data in accordance with these principles, the prospect is that the approach has the following advantages:

- **Overcoming Data Silos**: One can link information across applications, e.g. data from the Enterprise Resource Planning (ERP) systems with Customer Relationship Management (CRM) entries.
- **Unexpected Reuse**: Others can link their data to mine and vice versa –making it more useful, e.g. usage of data from the public Linked Data cloud like general information about farming in a country linked to the ebbits traceability scenario.
- **Independent Publication and Evolution of Data Structures**: There is no need for central control, data in Enterprise systems can be changed independently from data in the ebbits middleware.
- **Pay-as-you-go Data Integration**: There is no high up-front cost for integration, the most important links can be set first and more and more links can be added later on.
- **Leveraging Up-to-date, Externally Managed Information:** Information is linked to instead of copied, which prevents inconsistencies, saves data management costs and delivers up-to-date information instead of copying soon to be outdated information.

Of course, the usage of Linked Data has also some preconditions, also with regard to the ebbits architecture. This includes the control over the publication of data which has to be solved within the security framework of ebbits and should restrict the part of the data which is published in RDF. Other concerns, which are currently also areas of active research, include:

- **Inconsistent Data:** Due to the decentralized management, there might be inconsistencies in the data. The approach to circumvent this problem is the annotation of the data with provenance and quality meta-data (see also Section 4) and the use of appropriate evaluation mechanisms.

- **Performance Issues:** As current RDF stores are already able to handle very large amounts of data, the persistence should not be a great problem. Following links over the network will take time. Of course, this is a problem for any distributed system. Caching the data will speed up repeated accesses. For low-level sensor and actuator events, the solution could be the use of only aggregated values, i.e., the events are processed and aggregated at each local node and only those results of these computations, which are also interesting for higher level components and possibly enriched with semantic annotations, are used for the Linked Data approach.
- **Missing Established RDF Data Formats:** Work Package 4 should define common vocabularies and therefore establish data formats standards within the ebbits project. A starting point in this direction is also the Deliverable 3.2 about "Vertical and horizontal business vocabularies" (Sabol et al, 2011) from Work Package 3.

6.2 Publishing in RDF Stores and Database-to-RDF Converters

Linked Data is expressed in the RDF format. Therefore, the principles and technologies used in RDF stores and that have been described in Section 5.4 also hold for publishing Linked Data in such stores.

On the other hand, it is not always desirable to put all the data into an RDF store. If the information is already stored in a relational database and these data sets are updated very often in the database, it is better to use a so-called database-to-RDF converter to map the content of the database to RDF instead of extracting the data every time into an RDF store. One such system for publishing relational data on the web is the D2R Server (Bizer & Cyganiak, 2006). This tool enables RDF and HTML browsers to navigate the content of non-RDF databases and allow for querying the database using the SPARQL RDF query language. Requests for navigation and search from the web are translated on-the-fly to SQL queries for the database. In this way, even the content of large databases can be made available in the Linked Data format with acceptable response times.

In Figure 7, the general architecture of the D2R server is depicted. The data in the database is mapped using the customizable D2RQ mapping language. Such a mapping specifies how resources are identified and how property values can be constructed from database content (Bizer & Cyganiak, 2006). The D2R server also includes a tool which takes a look into the database, especially for table and column names, and automatically generates a first mapping from the table structure. This initial mapping can then be adjusted by the user. The D2R provides a Linked data interface for the mapped data. This interface makes RDF descriptions of individual resources available over the HTTP protocol which then can be browsed by an RDF browser (see also Section 6.5 below). SPARQL clients are enabled by the SPARQL interface to query the database in this protocol. Thereby, queries are executed against the complete database, i.e., a virtual RDF graph representation of it. For web browsers, which use HTML, another interface for this format is provided. Which interface and therefore also which data format is used for answering a request from the web is determined by content negotiation relying on Internet media types.

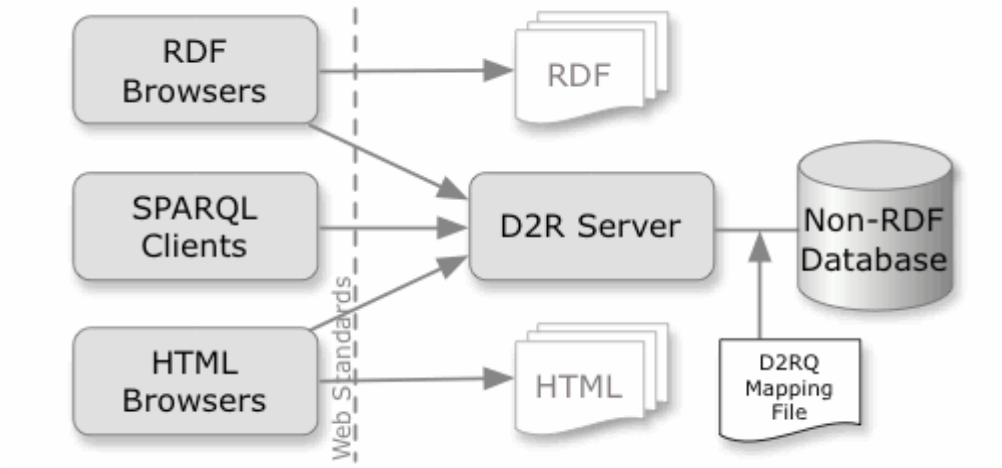


Figure 7: The architecture of the D2R Server, Source: (Bizer & Cyganiak, D2R Server - Publishing Relational Databases on the Semantic Web (Poster), 2006)

6.3 Link Generation

The mapping of the data and meta-data into the RDF format is just one step, the added value can only then be achieved if the data is interlinked between different data sources. For large data sources, this task must be undertaken automatically or at least semi-automatically. A tool that supports data publishers in setting RDF links to other data sources on the web is Silk - Link Discovery Framework (Volz, Bizer, Gaedke, & Kobilarov, 2009), which has the following main features:

- **Different Link Types**: It allows for the generation of owl:sameAs links, which indicates that two URI references actually refer to the same thing, as well as other types of RDF links.
- **Declarative Language**: It provides a flexible, declarative language with which developers can specify which types of RDF links should be discovered between data sources as well as which conditions data items must fulfil in order to be interlinked. These link conditions can depend on a combination of various similarity metrics and can take the graph around a data item into account. For the latter condition, Silk provides a simple RDF path selector language for providing parameter values to similarity metrics and transformation functions.
- **No Replication**: Silk accesses remote data sources via a SPARQL endpoint and therefore can be used in distributed environments without having to replicate the datasets locally. Furthermore, it is possible to restrict the query load on remote sources by setting a delay for subsequent queries.
- **Different Ontologies**: Silk can still be applied in situations where terms from different vocabularies are mixed and a unique consistent ontology does not exist.
- **Performance Improvements**: It employs various caching, indexing and entity preselection methods to increase performance and reduce network load.
- **Meta-data Annotation**: The links, which the Silk framework discovers, can be given out as simple RDF triples or in a reified form. In the latter case, the creation date and the confidence score together with an identification of the used heuristic can be attached to the resulting RDF link. These annotations could for example then be used in the ebbits project as meta-data about the quality of the links.

Silk is written in Python and its architecture is depicted in Figure 8. Silk starts by retrieving source and target resource lists (Volz, Bizer, Gaedke, & Kobilarov, 2009). The former is retrieved through a resource lister and cached on disk. The latter is first indexed by a resource indexer and only specific target resource candidates from the index are later used for comparison. For the actual comparison loop, user defined metrics are evaluated for each resource pair. If the metrics or functions contain RDF path values as parameters, these are resolved by an RDF Path Translator to SPARQL queries

and the queries are evaluated. All (intermediate) query results are cached in the RDF Path Memory Cache. If the comparison of a resource pair results in a value above a certain threshold, which itself is retrieved from the configuration file, and the maximum number of outgoing links has not been reached yet, the candidate link is written to the links output file.

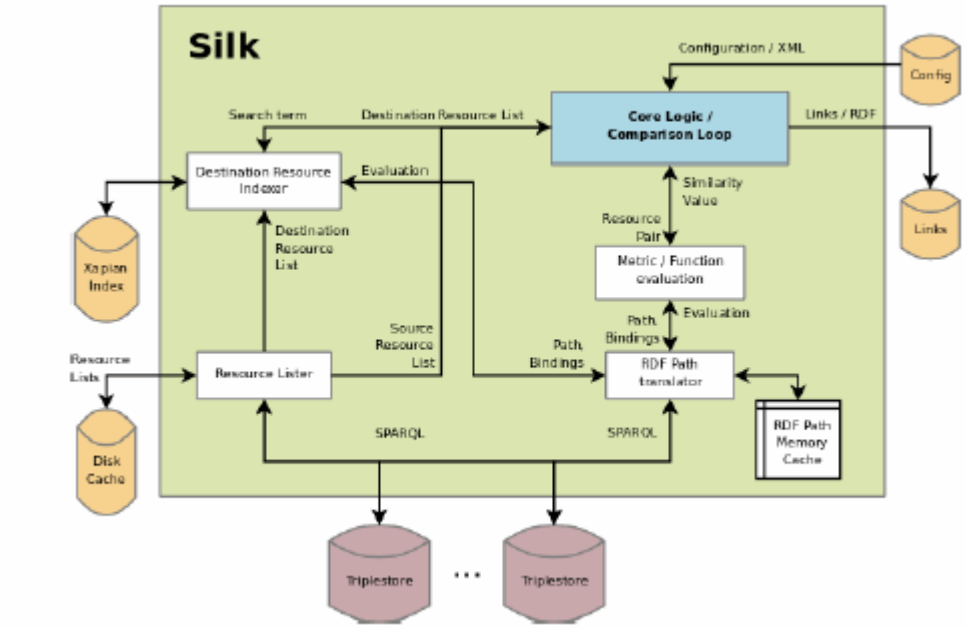
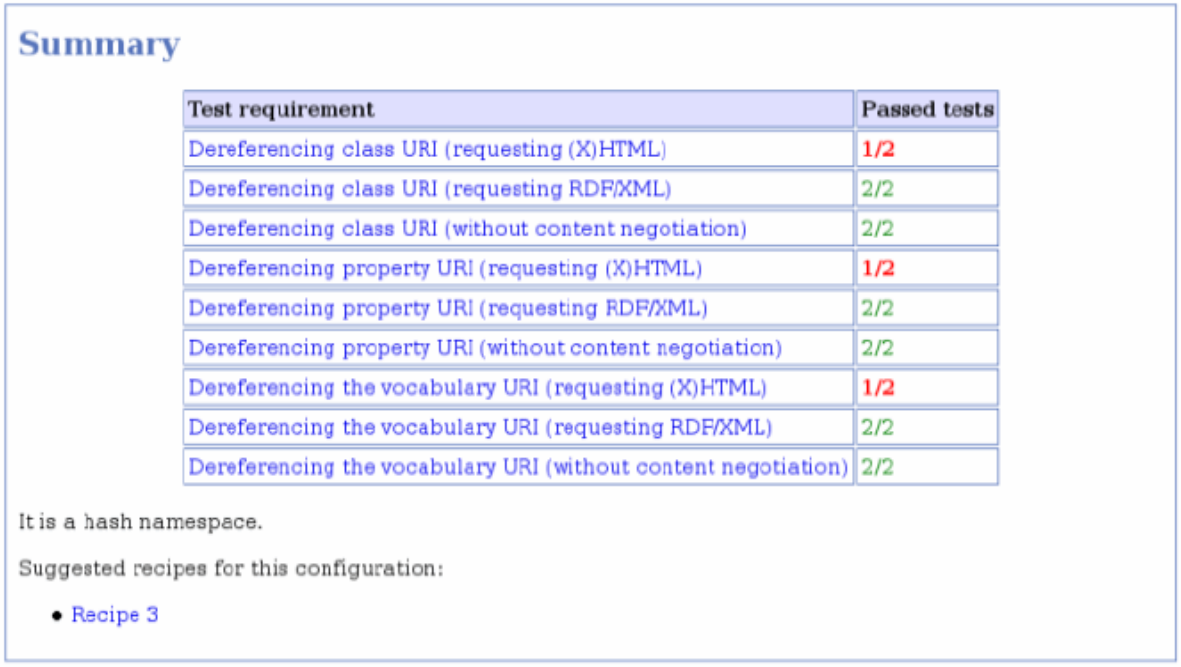


Figure 8: The Silk System Architecture, Source: (Volz, Bizer, Gaedke, & Kobilarov, 2009)

Data can also be linked if it is contained in the highly dynamic infrastructure of a P2P file-sharing network. In such an infrastructure, the links are based on a special URI scheme which allows unambiguous designation of replicated data items, regardless of their location in the network. For details see (Davoust & Esfandiari, 2009).

6.4 Validation

Once the data is represented in RDF and links are inserted between the different resources, it can be published on the web. Documents on the web are usually retrieved using the HTTP protocol. This protocol uses a technique called content negotiation. This mechanism allows serving web content in the format and language preferred by requester and subject to availability. Using this method, which is transparent for the user, has many benefits and it can be implemented in different ways on a web server, for example, the Apache web server (Berrueta, Fernández, & Frade, 2008). With the intended publication of RDF vocabularies this mechanism gets more complicated as there is now a new dimension of machine-readable versus human-readable data depending on the browser and the requesting user. To facilitate the task of testing the correct configurations for content negotiation on web servers with regard to RDF vocabularies, e.g., delivering RDF data for RDF browsers and html for normal browsers, a web-based application named Vapour was developed. This application is written in Python and provides a service that makes multiple requests to a set of URIs during the run of a test suite. The responses of the server are checked and in case of errors pointers to best practices are provided. Tests are run for the vocabulary URI, a class URI and a property URI (see Figure 9 for an example).



Summary

Test requirement	Passed tests
Dereferencing class URI (requesting (X)HTML)	1/2
Dereferencing class URI (requesting RDF/XML)	2/2
Dereferencing class URI (without content negotiation)	2/2
Dereferencing property URI (requesting (X)HTML)	1/2
Dereferencing property URI (requesting RDF/XML)	2/2
Dereferencing property URI (without content negotiation)	2/2
Dereferencing the vocabulary URI (requesting (X)HTML)	1/2
Dereferencing the vocabulary URI (requesting RDF/XML)	2/2
Dereferencing the vocabulary URI (without content negotiation)	2/2

It is a hash namespace.

Suggested recipes for this configuration:

- Recipe 3

Figure 9: Report summary made by Vapour, Source: (Berrueta, Fernández, & Frade, 2008)

The report of the Vapour tool is useful to debug and advice on how to solve common problems for content negotiation. Furthermore, as the result of the tests is also provided in the RDF format, it should also be easy to take these machine-readable data to build other services on top of Vapour (Berrueta, Fernández, & Frade, 2008).

6.5 RDF Browsers and Presentation to the User

Users should have the possibility to easily browse and navigate within the published Linked Data. A generic browser for these interlinked data is the Tabulator project (Berners-Lee et al., 2006). This project is domain-independent and is written in JavaScript. Furthermore, Tabulator runs on the user's machine and is therefore a decentralized design which allows for scalability with growing adoption. As Tabulator is a RDF browser, it navigates along relationships in a web of concepts, which is the conjunction of all the RDF graphs that have been read. This is in contrast to a "normal" web browser which browses along links in documents. Nevertheless, Tabulator also has an awareness of the underlying web of documents to always allow the user to check the provenance of data. Additionally, the user is informed about the necessity to access data remotely with its corresponding time delay and the unavailability of sources due to network failures.

Tabulator operates in two interlocking modes- exploration and analysis (Berners-Lee et al., 2006). The exploration mode corresponds to followings links in a normal web browser. The user starts by given Tabulator an URI and then can explore the RDF graph in a tree view by expanding nodes to get more data about them. Tabulator implicitly follows RDF links and retrieves more RDF data about the corresponding nodes. This link-following is not done, however, in an undirected traversal without limit, instead, only the information is loaded which was requested by the user. Exploring in Tabulator is done in the outliner mode, which is depicted in Figure 10 and which is a natural way for displaying tree-oriented data and many people are comfortable with it. Such a tree-oriented view also works for data with many nodes and many different properties, where for example circle-and-arrow representation of the RDF graph is no longer appropriate.

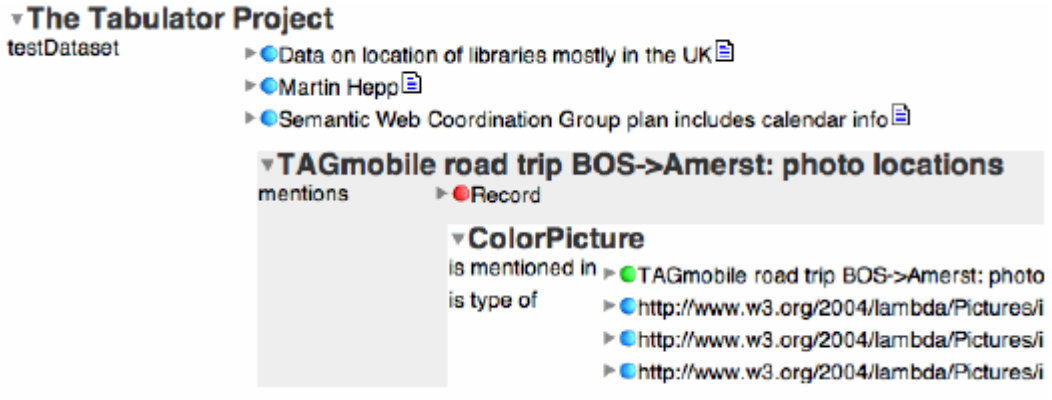


Figure 10: Exploring in Outliner View, Source: (Berners-Lee et al., 2006)

The analysis mode of Tabulator allows the user to select a predicate to define a pattern and let Tabulator perform a query to find all examples of that pattern (Berners-Lee et al., 2006). The query pattern is matched against the RDF graph and the results can be presented in various modular views, including tables. These views allow for the dense representation of data. The user can switch back to the exploration mode by double-clicking on any table cell. Tabulator has several such views to emphasize specific properties of RDF data. In Figure 11, there are two examples shown, the Calendar View and the Map View. The former enables the user to see when events overlap and visually assess event durations. The user can navigate between months with calendar data and each day's events are ordered by time. The latter can display multiple queries on a single view in order to compare geographical locations. The view plots RDF latitude and longitude data that is the result of a specific query. Additionally, it can handle addresses using the geocoding feature in the Google Maps API.

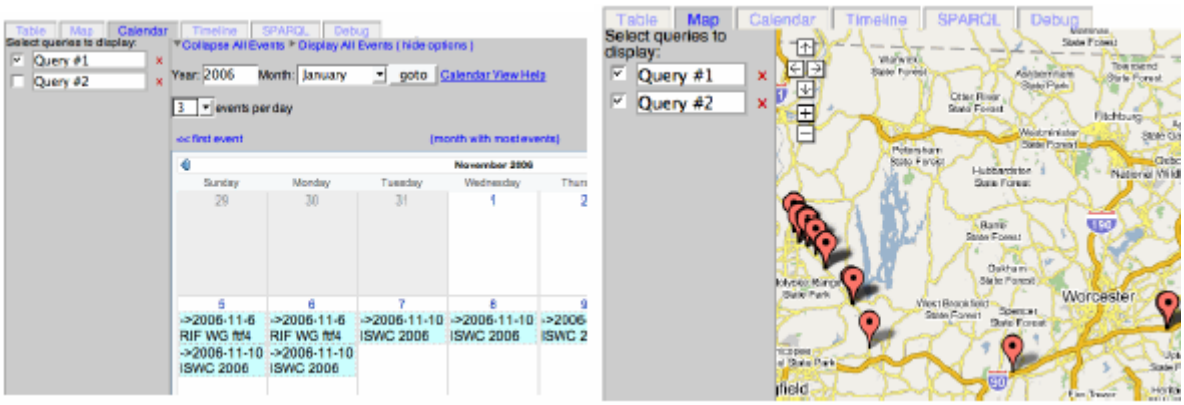


Figure 11: Analyzing query results in Calendar View and Map View, Source: (Berners-lee, et al., 2006)

RDF browsers like the above described Tabulator all have to solve the task of presenting content primarily intended for machine consumption in a human-readable way. Doing so requires a solution to two sub-problems: identifying what information contained in the RDF models should be presented, i.e., content selection, and how this data should be presented to users, that is, content formatting and styling. In order for not having to specify this kind of RDF presentation knowledge for every RDF browser but instead to share and reuse such knowledge across application and for different users, a browser-independent presentation vocabulary for RDF was created. It is called Fresnel. It is an RDF vocabulary, thus the presentation knowledge is expressed declaratively in RDF. The vocabulary itself is specified by an OWL ontology (Pietriga, Bizer, Karger, & Lee, 2006). It consists of a set of core presentation concepts that should be applicable across domains. Futhermore, the core modules of Fresnel should be easy to learn and use, but also easy to

implement to be adopted by many applications. On top of these core modules one can also define additional Fresnel vocabulary which then is restricted to only some domains.

Displaying all the data which is contained in an RDF model makes it difficult to extract information relevant to the current task and leads to cognitive overload for the user. Fresnel uses a foundational concept called lenses (see also Figure 12) to restrict the visualization to small but cohesive parts of the RDF graph. Thereby, lenses specify which properties of RDF resources are shown and they also determine the order of these properties. The second foundational concept is called formats. These formats indicate how the previously selected content can be laid out properly and rendered with graphical attributes that lowers the cognitive effort of the user and facilitates general understanding of the displayed information. To achieve this, formats indicate how to format content selected by lenses and may also generate additional static content and hooks that can be used to style the output through external CSS style sheet.

For further details please see (Pietriga, Bizer, Karger, & Lee, 2006).

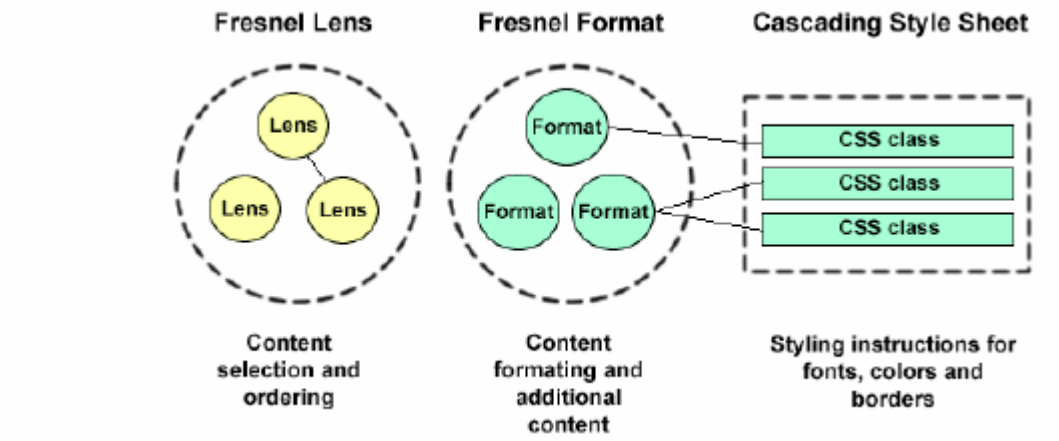


Figure 12: The Foundational Concepts of Fresnel, Source: (Pietriga, Bizer, Karger, & Lee, 2006)

7. Conclusions

This deliverable reports on the state-of-the-art of systems with persistent and high volume data with regards to the ebbits architecture. Due to the ontological support and commitments within the ebbits project and its very distributed architecture, this report focuses on persistence of high volumes of RDF data and the propagation of these data using the Linked Data approach. In addition to that, there are many different categories of meta-data that with potential use in the ebbits architecture and in its applications, some of these are exemplified in this deliverable.

The representation of data and meta-data in RDF and its persistent storage in high volumes is described in this report. Furthermore, a new approach to interlink data and create added value is presented and aspects for creating RDF data out of relational databases, the automated link generation and the presentation to the user are reported in detail among other technologies.

The technologies, mechanisms, tools, trends identified in this deliverable will form a basis for additional work in Work Package 6. Besides further work in Task 6.1 "Investigation and enhancement of propagation mechanisms for ebbits data", they can support the matching between different information representations in Task 6.2 "Investigation and enhancement of matching services between platform and enterprise systems". Furthermore, it can shore up the handling of knowledge enriched data with the meta-data for the enhancement of interaction mechanisms between Business Intelligence functionality in Enterprise Systems and the Distributed Intelligence to be provided by the ebbits platform, which is the content of Task 6.3 "Investigation and enhancement of integration mechanisms for distributed intelligence and business intelligence".

8. References

Ahlsen, M., Kool, P., Kostelnik, P. and Rosengren, P. (2010). "D6.11 Hydra SOA and Model-Driven Architecture, Final Report." Hydra Project Deliverable, Rep. No: D6.11, IST project 2005-034891, Oktober 2010.

Alcaraz, G., Cultrona, P. A., Duran, M. C., Kool, P., Pastrone, C., Spirito, M. and Tomasi, R. (2011). "D8.4 Integration of physical world in manufacturing." ebbits Project Deliverable, Rep. No: D8.4, FP7 project 257852 - ebbits, June 2011.

Berners-Lee, T. (2006). *Linked Data*. Retrieved 2011, from http://www.w3.org/DesignIssues/LinkedData.html

Berners-Lee, T. (2005). Uniform Resource Identifier (URI): Generic Syntax. *Memo* .

Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., et al. (2006). Tabulator: Exploring and analyzing linked data on the semantic web . *3rd International Semantic Web User Interaction Workshop .* Athens, Georgia.

Berrueta, D., Fernández, S., & Frade, I. (2008). Cooking HTTP content negotiation with Vapour. *4th workshop on Scripting for the Semantic Web.* Tenerife, Spain.

Bizer, C., & Carroll, J. (2004). Modelling Context using Named Graphs. *SWIG Meeting.* Cannes.

Bizer, C., & Cyganiak, R. (2006). D2R Server - Publishing Relational Databases on the Semantic Web (Poster). *5th International Semantic Web Conference .* Athens, Georgia .

Checcozzo, R., Glova, J., Jacobsen, M., Sabol, T. and Sabol, T. (2011). "D3.1 Enterprise Use Case." ebbits Project Deliverable, Rep. No: D3.1, FP7 project 257852 - ebbits, Feb 2011.

Davoust, A., & Esfandiari, B. (2009). Linking and Navigating Data in a P2P FileSharing. *Linked Data on the Web Workshop .* Madrid, Spain: CEUR Workshop Proceedings.

Furdik, K., Knechtel, M., Tomasi R., Caceres M., Kostelnik, P., Hreno, J. (2011). D4.3 Coverage and scope definition of a semantic knowledge model ." ebbits Project Deliverable, Rep. No: D4.3, FP7 project 257852 - ebbits, May 2011.

Gustafsson, A. and Sörman, U. (2004). "Data Quality Management. A way to control costs in an organisation (In Swedish)." Examensarbete, Rep. No: C04/2004, Högskolan Dalarna & Vägverket 2004.

Hertel, A., Broekstra, J., & Stuckenschmidt, H. (2008). RDF Storage and Retrieval Systems.

Hitzler P., Krötzsch M., Rudolph, S., Sure, Y. (2008). *Semantic Web – Grundlagen.* Springer-Verlag Berlin Heidelberg.

IEC/ISO (2003). "IEC 62264-1 Enterprise-control system integration - Part 1: Models and terminology." Internaional standard, IEC 62264-1:2003(E), IEC/ISO, 2003-03.

IEC/ISO (2004). "IEC 62264-2 Enterprise-control system integration - Part 2: Object model attributes." International Standard, IEC 62264-2:2004(E), IEC/ISO, 2004-07.

ISO (2009). "Information and documentation -- The Dublin Core metadata element set ", International Standard, ISO 15836:2009, ISO, Feb 2009.

Knechtel, M., Hreno, J., Pramudianto, F., Ahlsen, M., Furdik, K. (2011). D4.1 Analysis of Semantic Stores and Specific ebbits Use Cases. ." ebbits Project Deliverable, Rep. No: D4.1, FP7 project 257852 - ebbits, February 2011.

Kool, P., Ahlsén, M., Alcaraz, G., Björnsson, S., Franceschinis, M., khaleel, H., Pastrone, C. and Vinkovits, M. (2011). "D8.1 ebbits network architecture." ebbits Project Deliverable, Rep. No: D8.1, FP7 project 257852 - ebbits, Aug 2011.

Kostelnik, P., Hreno, J., Ahlsen, M., Alcaraz, G., Pastrone, C., Spirito, M., Madsen, T. N., Checcozzo, R., Paralič, M. and Furdík, K. (2011). "D7.2 Event and data structures, taxonomies and ontologies." ebbits Project Deliverable, Rep. No: D7.2, FP7 project 257852 - ebbits, February 2011.

McBride, B. (2001). Jena: Implementing the RDF Model and Syntax Specification. *in Proceedings of the 2001 Semantic Web Workshop.* Hong Kong, China.

Pietriga, E., Bizer, C., Karger, D., & Lee, R. (2006). Fresnel: A Browser-Independent Presentation Vocabulary for RDF. *5th International Semantic Web Conference .* Athens, Georgia .

Sabol, T., Glova, J., Nutakki, P., Knechtel, M., Cultrona, P., Madsen, T. N., Jacobsen, M. and Björnsson, S. (2011). "D3.2 Vertical and horizontal business vocabularies." ebbits Project Deliverable, Rep. No: D3.2, FP7 project 257852 - ebbits, Aug 2011.

Volz, J., Bizer, C., Gaedke, M., & Kobilarov, G. (2009). Silk - A Link Discovery Framework for the Web of Data. *WWW2009 Workshop on Linked Data on the Web.* Madrid, Spain: CEUR Workshop Proceedings.

W3C. (2011). *Naming and Addressing: URIs, URLs*. Retrieved from http://www.w3.org/Addressing/

W3C (2007). "SAWSDL - Semantic Annotations for WSDL and XML Schema." W3C Recommendation, Rep. No: www.w3.org/TR/2007/REC-sawsdl-20070828/ W3C, W3C, 2007-08-27.