Learning Applications based on Semantic Web Technologies

MATTHIAS PALMÉR

Doctoral thesis
Stockholm, Sweden 2012
Akademisk avhandling som med tillstånd av Kungliga Tekniska Högskolan framlägges till offentlig granskning för avläggande av teknologi doktorsexamen i medieteknik torsdagen den 22 november 2012 klockan 10.00 i sal F3, Lindstedsvägen 26, KTH Campus, Stockholm.

© Matthias Palmér, 2012
Abstract

The interplay between learning and technology is a growing field that is often referred to as Technology Enhanced Learning (TEL). Within this context, learning applications are software components that are useful for learning purposes, such as textbook replacements, information gathering tools, communication and collaboration tools, knowledge modeling tools, rich lab environments that allows experiments etc. When developing learning applications, the choice of technology depends on many factors. For instance, who and how many the intended end-users are, if there are requirements to support in-application collaboration, platform restrictions, the expertise of the developers, requirements to inter-operate with other systems or applications etc.

This thesis provides guidance on a how to develop learning applications based on Semantic Web technology. The focus on Semantic Web technology is due to its basic design that allows expression of knowledge at the web scale. It also allows keeping track of who said what, providing subjective expressions in parallel with more authoritative knowledge sources. The intended readers of this thesis include practitioners such as software architects and developers as well as researchers in TEL and other related fields.

The empirical part of the this thesis is the experience from the design and development of two learning applications and two supporting frameworks. The first learning application is the web application Confolio/EntryScape which allows users to collect files and online material into personal and shared portfolios. The second learning application is the desktop application Conzilla, which provides a way to create and navigate a landscape of interconnected concepts. Based upon the experience of design and development as well as on more theoretical considerations outlined in this thesis, three major obstacles have been identified:

The first obstacle is: lack of non-expert and user friendly solutions for presenting and editing Semantic Web data that is not hard-coded to use a specific vocabulary. The thesis presents five categories of tools that support editing and presentation of RDF. The thesis also discusses a concrete software solution together with a list of the most important features that have crystallized during six major iterations of development.

The second obstacle is: lack of solutions that can handle both private and collaborative management of resources together with related Semantic Web data. The thesis presents five requirements for a reusable read/write RDF framework and a concrete software solution that fulfills these requirements. A list of features that have appeared during four major iterations of development is also presented.

The third obstacle is: lack of recommendations for how to build learning applications based on Semantic Web technology. The thesis presents seven recommendations in terms of architectures, technologies, frameworks, and type of application to focus on.

In addition, as part of the preparatory work to overcome the three obstacles, the thesis also presents a categorization of applications and a derivation of the relations between standards, technologies and application types.
I want to start by thanking my supervisor Ambjörn Naeve for his support over the years. His ideas and visions, although at times a bit hard to reach or realize, have always been inspiring and provided a purpose for the work we have done together. I also want to thank my other supervisors over the years, initially Yngve Sundblad and lately Marko Turpeinen for providing the structured guidance that have been so important to be able to finalize this thesis.

My former colleague and long time friend Mikael Nilsson have been an unending source of good advice and interesting discussions. The work we have done together have been truly fun and exhilarating and I hope we will get the opportunity to work together in the future. I also deeply appreciate the collaboration with other colleagues in the research group: Fredrik Enoksson for his persistence, attention to detail and his never ending pop-cultural references that I sometimes have to google to understand. Hannes Ebner for his knowledge on a broad range of technologies, for acting as an information filter by going through hundreds of news feeds daily and for being a fellow skeptic. Erik Isaksson for the stringency in the discussions which have helped me to sharpen the argumentation in certain areas of the thesis. And finally for Fredrik Paulsson for his brutal honesty from afar.

I also appreciate my colleagues at Uppsala Learning Lab as you have all become good friends and the numerous times when you have pushed me forward by curiously inquiring about my progress. I am especially grateful for the support and understanding from Mia Lindegren who has allowed me to complete my thesis in parallel to other tasks. A special thanks goes to Henrik Eriksson, Jöran Stark and Jan Danils who have all spent so much time developing software that this thesis in part reports upon.

On a more personal note, I want to thank my parents for their support and acceptance of my sometimes geeky behavior during my upbringing. I appreciate all the lively discussions we have had, they have helped me to form a critical attitude. I am also grateful for the resistance my sisters put up when wanted to share my fascination with math. Realizing that interest in a topic does not necessarily follow from facts or logic gave me both an interest and a hint of the challenges of the learning process. I also want to thank my relative Maria for always being there, listening and providing valuable feedback to my inquires about life and philosophy.

I am very grateful for my children, Vilmer and Alma, who have allowed me, from time to time, to drop the adult character and enjoy the simple act of childs play again. They have also laboriously taught me patience and humility in the uniqueness of life. But most important, I want to thank Kajsa, my loving partner, for appreciating me for who I am and supporting me all the way through this endeavor. I love you and our children dearly.

Finally, I want to mention my grandfather Affe (Alfred Palmér) who early in my life gave me the curiosity and the feeling that anything was possible. The days we spent tinkering in the old school transformed into workshop, taking things apart, building new things from steal and wood or doing math at the kitchen table have been fundamental in forming who I am today. These days were the best learning experience I have ever had, although there was no teaching, only doing. Affe, you are long gone by now, but I will try to pass your enthusiasm and curiosity to the next generation. Also, with this thesis I am trying to take small steps towards more interesting learning applications that can eventually support learning by doing and including mentors in the process.
Contents

Abstract........................................................................................................................................... 1
Acknowledgements.......................................................................................................................... 2
Included Papers.................................................................................................................................. 6

1. Introduction.................................................................................................................................... 1
   1.1 The Purpose of this Thesis........................................................................................................ 2
   1.2 Problem Definition.................................................................................................................. 2
   1.3 Methodology........................................................................................................................... 3
   1.4 Terminology................................................................................................................................ 4
      1.4.1 Generic terms..................................................................................................................... 4
      1.4.2 Acronyms.......................................................................................................................... 4
      1.4.3 Terminology of Developed Software.................................................................................. 6
   1.5 How to read this Thesis........................................................................................................... 6

2. Using Semantic Web for Learning Applications............................................................................. 9
   2.1 Perspectives on Learning........................................................................................................ 9
   2.2 Semantic Web Basics............................................................................................................ 10
   2.3 RDF Supports Objective Metadata....................................................................................... 12
   2.4 RDF Supports Subjective Metadata..................................................................................... 13
   2.5 RDF Supports an Evolving Human Discourse..................................................................... 14
   2.6 The Semantic Web and Design Principles for Learning Applications............................... 14
   2.7 The Implicit Requirements of the Semantic Web................................................................... 20
   2.8 Summary.................................................................................................................................. 20

3. Architectures, Technologies and Application Types....................................................................... 21
   3.1 Web Architecture.................................................................................................................... 21
   3.2 Semantic Web Data................................................................................................................. 23
   3.3 Service-Oriented Architecture............................................................................................... 24
   3.4 REST......................................................................................................................................... 25
   3.5 Resource Oriented Architecture.............................................................................................. 27
   3.6 Linked Data.............................................................................................................................. 28
   3.7 Applications Types.................................................................................................................... 29
   3.8 Summary.................................................................................................................................. 32

4. Two Semantic Web based Learning Applications...................................................................... 35
   4.1 EntryScape - a Personal and Collaborative Portfolio Suite.................................................. 35
      4.1.1 The purpose of EntryScape............................................................................................... 35
      4.1.2 How EntryScape works.................................................................................................... 36
      4.1.3 Added value of Semantic Web.......................................................................................... 39
4.2 Conzilla - a Concept Browser .................................................................40
  4.2.1 The purpose of Conzilla ..................................................................40
  4.2.2 How Conceptual Browsing and Conzilla Works .........................41
  4.2.3 Added value of Semantic Web .........................................................44
4.3 Identifying major Obstacles .................................................................45
  4.3.1 Obstacle 1 ......................................................................................45
  4.3.2 Obstacle 2 ......................................................................................46
  4.3.3 Obstacle 3 ......................................................................................46
5. Presenting and Editing RDF .................................................................47
  5.1 Tool Categories for Presenting and Editing RDF .........................47
  5.2 Presenting and Editing RDF in Graph Based Interfaces .............49
  5.3 Presenting and Editing RDF in Forms .............................................50
  5.4 Related Initiatives for Configurable RDF Forms .........................51
  5.5 Six Iterations Towards Configurable RDF Forms .......................53
    5.5.1 SCAM Portfolio Metadata Editor version 1 .........................53
    5.5.2 The SHAME library .................................................................54
    5.5.3 SCAM Portfolio Metadata Editor version 2 .........................56
    5.5.4 SHAME 2 ..................................................................................57
    5.5.5 Ajax-SHAME .............................................................................58
    5.5.6 RForms .....................................................................................59
  5.6 Summary .........................................................................................61
6. Read/Write Resource and RDF Framework .....................................63
  6.1 Breaking Down the Second Obstacle ..........................................63
  6.2 Existing Solutions ...........................................................................64
  6.3 Four iterations .................................................................................66
    6.3.1 SCAM 1 - efolio ..........................................................................66
    6.3.2 SCAM 2 .....................................................................................67
    6.3.3 SCAM 3 .....................................................................................69
    6.3.4 SCAM 4 - also known as EntryStore .......................................71
  6.4 Summary .........................................................................................73
7. Recommendations ...............................................................................75
  7.1 Rely on the Web Architecture ......................................................75
  7.2 Commit to REST and get guidance from ROA ..............................76
  7.3 Expose your information as Linked Data ....................................76
  7.4 Base your application on a read/write RDF framework ..............76
  7.5 Build Web Applications first .......................................................77
  7.6 Build RESTful Ajax Web Applications .......................................77
  7.7 Use a framework to help you present and edit RDF ..................77
8. Conclusions .......................................................................................79
  8.1 Research questions .........................................................................80
  8.2 Contributions of this thesis ............................................................82
  8.3 Future Work ...................................................................................82
9. Summary of Papers

9.1 Paper 1: E-Learning in the Semantic Age


9.3 Paper 3: The SCAM framework - helping semantic web applications to store and access metadata

9.4 Paper 4: Conzilla – a Conceptual Interface to the Semantic Web

9.5 Paper 5: Annotation profiles: Configuring forms to edit RDF


References

Papers
Included Papers

Paper 1

Paper 2

Paper 3

Paper 4

Paper 5

Paper 6

A summary of each paper and the papers themselves can be found at the end of this thesis.
1. Introduction

Traditional learning environments, including most formal learning environments at schools and universities, support information flowing from the teacher to the students. Even though there are other didactic methods, the basic lecture style is still to read and write before the students. The students then normally copy as much as possible of what they hear and see. The course usually ends with a test that shows that the students have learned the presented material - or perhaps just learned to mimic.

The first incarnation of the web was very much like traditional teaching. A few people/companies/organizations were producing content and many others were consuming it. However, changes of attitudes toward web-usage, as well as improvements in web tools and technologies have lead to drastic changes in behavior. This development is often referred to as Web2.0. In contrast to merely being passive web-consumers, many people are now actively participating in social networks, creating and providing content as well as comments on what others have contributed. At one end of the spectrum, this corresponds to the day-to-day communication between people that have moved into the digital domain. Moreover, this communication has been scaled up to a to potentially global arena. At the other end, the traditional human discourses between privileged scholars, intellectuals, journalists, and politicians have now been democratized.

There are no reasons why traditional learning environments would not be affected by this development. In fact, the online community is driving an evolving discourse, by constantly creating new material, as well as commenting and communicating around a vast array of topics. There is a need to reap the benefits of this discourse for learning purposes and combine the new learning activities of the online community with the traditional learning activities that still need to happen in the classroom. Therefore, a change is needed in both pedagogy and technology. From the pedagogical perspective this includes shifting the role of the teacher from the traditional knowledge preacher/filter, towards more of a knowledge coach (Naeve, 2001a). From the technical perspective, there is a need for tools and technologies that can support more complex discourses in a way that creates overview while preserving depth and context. This complexity increases the need for multiple interpretations to coexist without forcing consensus.
Semantics is about interpretation, which is necessary to create shared understanding. Hence, it is easy to see the potential capacity of Semantic Web technologies for supporting a global network of teachers and learners in such interpretative activities.

This thesis discusses the benefits of using Semantic Web technology for building learning applications. It focuses on the technical perspective, more specifically on how various aspects of Semantic Web technology can enable people to express themselves and communicate with increased precision on a growing and changing range of topics. It identifies three important obstacles to using Semantic Web for building learning applications. The thesis also addresses the identified obstacles by showing how they can be overcome in practice.

The obstacles have been investigated using two different application domains for experiments. One is e-portfolios that aim to help individuals and groups to create and organize a wide range of resources, including online web resources and uploaded files such as learning material, comments, reflections, formal competency descriptions etc. The other problem domain is concept browsing (Naeve, 2001b)(Naeve, 1999) which aims to support the exploration, expression, communication and collaboration around concepts and their relations.

1.1 The Purpose of this Thesis

The starting point is a vision rather than a problem statement. This vision can be expressed as:

Learning applications based on Semantic Web technologies will allow people to express themselves and communicate with increased precision on a growing and changing range of topics.

The vision has been one of the main motivators for the research presented in this thesis. Of course, it is neither disproved nor conclusively supported by the work of this thesis, although, section 2 provides a discussion of the benefits of focusing on Semantic Web technology for learning applications. Inspired by this vision the main purpose of this thesis is therefore:

To provide guidance for development of learning applications based on Semantic Web technologies.

The main target groups are practitioners such as software architects and developers but also researchers in fields such as TEL (Technology Enhanced Learning) and CSCL (Computer Supported Collaborative Learning).

1.2 Problem Definition

From the main purpose the following two research questions are derived:

I. What are the main obstacles when building Learning applications based on Semantic Web technologies?

II. How can these obstacles be overcome by using state-of-the-art web technologies and platforms?
The obstacles identified by the first question are summarized below, since they set the scene for the organization of the rest of the thesis. They are motivated by the long-term practical experience of developing learning applications. Two of these applications are introduced in chapter 4, where the obstacles they highlighted are identified and discussed in more detail. In summary, the obstacles are:

1. Lack of non-expert and user friendly solutions for presenting and editing Semantic Web data that is not hard-coded to use a specific vocabulary.

2. Lack of solutions that can handle both private and collaborative management of resources together with related Semantic Web data.

3. Lack of recommendations for how to build learning applications based on Semantic Web technology.

Of course, this list is not complete, and the concrete obstacles depend on the problem domain. Hence, in other problem domains, new obstacles may surface and one or several of the ones listed above may turn out to be unproblematic.

The second question, how to overcome the obstacles, is addressed throughout the thesis by a combination of experiences from practical development as well as from more theoretical concerns that are raised and discussed in this thesis and its included papers.

1.3 Methodology

The author has participated in the design and development of several learning applications of which two are discussed in section 4. During the development of these learning applications, the three obstacles introduced above have crystallized over time. Despite the fact that the obstacles have been derived from needs in specific learning applications this thesis argues that they have a wider relevance. The author has identified, refined and tried to find solutions to overcome the obstacles through an iterative process of research and development.

The first two obstacles are very concrete as they correspond to lack of solutions to concrete problems. In this case the author has been part of an iterative process to design and develop solutions for the obstacles. Each iteration corresponds to a more or less mature phase of the solution that has been deployed and undergone some form of testing. Since none of the solutions are standalone learning applications but rather constitute reusable parts of other learning applications, the author has not conducted separate field evaluations of the systems with a wide range of different users.

However, qualitative feedback from real users and/or developers are discussed in the analysis that is provided for each iteration. The analysis also contains discussions on software architecture including ideas and principles, major shifts in development, and similarities and differences with respect to other initiatives when they exist.

The last obstacle considers a lack of recommendations for how to develop Semantic Web based learning applications. During the work on this thesis the author has gradually refined and sometimes changed his position on what he believes should be considered good recommendations based on both theoretical concerns and practical experience. One example is how the final recommendations differ from what was recommended in the first paper. Several of the changes in beliefs are also reflected in the above mentioned iterations.
This thesis is based on work performed within several different projects from 2001 to 2011. Some of the projects have been EC-financed research projects such as ProLearn, LUISA and ROLE, while others have been based on collaboration with industry/authorities partners such as the Swedish Educational Broadcasting company, and the Swedish National Agency for Education which have resulted in technology being used in real world situations. There have also been funding from Wallenberg Global Learning Network (WGLN), Vinnova, as well as some support from Royal Institute of Technology (KTH), Uppsala University (more specifically Uppsala Learning Lab) and Umeå University.

Finally, some of the iterations have corresponded to more fundamental changes which have led to peer reviewed publications. The most important publications where the author has contributed a major part are the papers upon which this thesis is based.

1.4 Terminology

This section acts as a reference for the reader. First, it provides explanations of the generic terms used. Second, it explains acronyms that are used in more than one place. Third, it provides a terminology and short overview on the software upon which the experimental part of this thesis relies.

1.4.1 Generic terms

**Application**

A piece of software that includes a graphical user interface that provides a way for people to perform tasks.

**Web application**

An application that utilizes web technologies and runs in a web browser that resides either on a desktop or on a mobile device.

**Learning application**

An application intended to be used primarily to perform tasks that benefit learning.

**Resource**

Anything that may be identified by a URI. This includes web pages as well as abstract things, such as the concept of a circle, or physical objects, such as persons or cars.

**Information resource**

A resource such as a web page, the essential characteristics of which can be conveyed in a message.

**Data**

A piece of information that can be handled by a piece of software that knows about the syntax used.

**Metadata**

A piece of data that is intended to be interpreted as statements about a resource.

1.4.2 Acronyms
Ajax  *Asynchronous JavaScript and XML* - a group of technologies used to realize web applications that provide functionality without necessitating page loads. The definition of the term is no longer accurate, since the underlying requests for data are required neither to be asynchronous - nor to use XML as a data-format.

CSS  *Cascading Style Sheets* - a language to describes the look and formatting of documents that are written in markup languages such as HTML and XML.

DSP  *Description Set Profile* - a machine-processable expression for specifying which metadata terms to use (Nilsson, Miles, Johnston, & Enoksson, 2007).

HTML  *HyperText Markup Language* - a markup language used for expressing web pages.


LOM  *Learning Object Metadata* - an IEEE metadata standard for describing so called learning objects, which is the name given to resources that are intended to be used for learning.

JSON  *JavaScript Object Notation* - a light-weight data-interchange format, which is based on the object-literal expression in JavaScript.

JSP  *JavaScript Server Pages* - a templating mechanism in Java, often used to simplify the generation of HTML pages.

LMS  *Learning Management System* - a software application that is intended for the administration, documenting, tracking, reporting, and delivery of learning.

OWL  *Web Ontology Language* - a language based on RDF, which is used to formally describe ontologies, that is, classes, instances and their relations.

QEL  *Edutella Query Language* - a logic-based query language developed for the Edutella p2p network. QEL is expressed in RDF. See (Nejdl et al., 2002).

RDF  *Resource Description Framework* - a framework for representing information in the Web. RDF provides a formal semantics for expressing facts about resources.

RDFa  RDFa provides attributes that allow RDF triples to be embedded into HTML.

RDFS  *RDF Vocabulary Description Language 1.0: RDF Schema* - a language for defining vocabularies in RDF. RDFS is less expressive than OWL.

REST  *Representational State Transfer* - an architectural style describing the web. REST was introduced by Roy Fielding in (Fielding, 2000).

ROA  *Resource Oriented Architecture* - a software architecture that complies with REST. ROA was introduced by (Richardson & Ruby, 2007).

RPC  *Remote Procedure Call* - a way to request services from a computer program over a network via some protocol. The RPC approach is common in SOA.

SOA  *Service Oriented Architecture* - an architecture where software components are developed as independent services to facilitate interoperability and loose coupling.

SOAP  *Simple Object Access Protocol* - a protocol often used to realize SOA on the web.
1. Introduction

**SPARQL** SPARQL Query Language for RDF - a W3C-recommended query language for searching after information in RDF graphs.

**URI** Uniform Resource Identifier - an Internet Official Protocol Standard for describing global identifiers of resources.

**URL** Uniform Resource Locator - a subset of URIs that, in addition to identifying resources, also provides information about how to locate them.

**XForms** XForms - a markup language for generating forms that can edit XML instances.

**XML** Extensible Markup Language - a document-centric markup language for encoding a wide variety of data structures.

**XMPP** Extensible Messaging and Presence Protocol - a protocol for real-time communication that supports instant messaging, multi-part chat, presence, etc.

1.4.3 Terminology of Developed Software

**Conzilla** a learning application in the form of a Concept browser, where Context-maps containing concepts and concept-relations are navigated. *Development started 1999.*

**Confolio** a learning application in the form of an ePortfolio allowing uploaded files and linked to web resources to be described with metadata. *Development started 2008.*

**EntryScape** another name for Confolio.

**SCAM portfolio** an earlier version of EntryScape. *Development started 2001.*

**RForms** RDF Forms - a framework for presenting and editing metadata. *Development started 2010.*

**SHAME** Standardized Hyper Adaptable Metadata Editor - earlier version of RForms. *Development started 2002.*

**EntryStore** a framework for storing resources and their metadata (originally referred to as SCAM 4). *Development started 2008.*


1.5 How to read this Thesis

This thesis contains three major parts; (i) introduction and overview, (ii) formulation and discussion of the research questions analyzed, and (iii) the papers upon which the thesis is based.

The first part (i) contains chapter 1-3, where chapter 1 is what you are reading now, that is, an overall introduction, purpose, problems, methodology etc. Chapter 2 gives an introduction to the core Semantic Web technologies and describes how some of their characteristics can be
beneficial for learning applications. Chapter 3 provides a background as well as a discussion of how architectures and technologies are related to Semantic Web technology and various types of applications.

The second part (ii) corresponds to chapter 4-8, where chapter 4 introduces two learning applications and answers the first research question by identifying major obstacles. Chapter 5 addresses the first obstacle by discussing how to present and edit RDF both by looking at existing solutions and by going through six iterations of development of the SHAME/RForms framework. Chapter 6 addresses the second obstacle dealing with how to read and write RDF and resources first by breaking down the obstacle and looking at existing solutions and second by going through four iterations of the SCAM/EntryStore framework. Chapter 7 addresses the third obstacle by listing seven recommendations based on theory and practice including the work on the learning applications and the work done to overcome the obstacles. Chapter 8 provides conclusions and future work.

The third part (iii) contains the six papers on which this thesis is based, as well as a short introduction to each paper, including information about where it was published, the author's contributions etc.
2. Using Semantic Web for Learning Applications

As discussed in the introduction, the main focus of this thesis is how to build learning applications based on Semantic Web technologies. Not to motivate why Semantic Web technologies provide a solid base for building learning applications. In spite of this, it is worthwhile to briefly describe some of the added values for learning applications that are inherent in this approach. In order to do so we need to understand:

1. what the Semantic Web is about and what kind of useful characteristics it has.
2. how present and potential characteristics of learning application are related to different perspectives on learning that are inherent in some of the major educational theories.

Hence, this chapter starts with a brief overview of educational theories. Then, we go through the basics of the underlying technologies of Semantic Web. After that we explore a bit deeper by discussing three important aspects of the Semantic Web: objective metadata, subjective metadata, and the support for evolving human discourses. An attempt is made to understand the nature of learning applications from the perspective of these educational theories, and to identify some of the added values of Semantic Web technologies when building learning applications. Thereafter, we take a brief look at the implicit requirements of the Semantic Web. The chapter ends with a summary.

2.1 Perspectives on Learning

To attempt a general investigation of how educational theories relate to learning applications would be a formidable undertaking. Even restricting our attention to the most dominant educational theories would be out of scope for this thesis, since it has a largely technical focus. Instead we concentrate on certain perspectives of educational theories that share a common ground.
According to (Greeno et al., 1996) educational theories can be divided into three perspectives based on the way you look at knowledge and learning. Including only the learning aspect\(^1\), these perspectives can be described as:

**Behaviorist/empiricist** - learning is a process of forming and strengthening associations among mental or behavioral units. Feedback, intended to strengthen certain associations, is given to the learner based on the outcome of learning activities. The activities are often organized in a logical manner, from simple to more complex units of behavior.

**Cognitive/rationalist** - learning is about understanding concepts and theories through mental processes. The important sub-area of constructivism considers learning to be a form of understanding aided by an active process of construction rather than a passive assimilation of information. The learner is supposed to work with concrete material that can be manipulated to illustrate conceptual principles. Misconceptions can be useful stepping stones towards gradually becoming more attuned to the characteristics of the domain. General cognitive abilities such as reasoning, planning, and language comprehending are vital and complementary to more domain-specific approaches to solving problems.

**Situative/pragmatist-sociohistoric** - learning is about getting accustomed to, and improving, the successful participation of the learner in the social activities of a community. Learners start as beginners and develop their knowledge by actively participating in the community, although their activities are often initially somewhat peripheral. Over time, when learners progresses, their actions are perceived more and more central to the community and begin to add to the practice. It is argued that a learner's identity is in part formed by active and successful participation in a community of practice. Hence, becoming a more central participant of a community can be a powerful motivation for learning.

(Greeno et al., 1996) also states at page 16 that the perspectives "help to frame theoretical and practical issues in distinctive and complementary ways". Therefore, it is likely that when developing learning environments/applications, it would make sense to simultaneously consider several of these perspectives.

The perspectives introduced are not unique and others have chosen different discriminating dimensions for the categorization of educational theories. For instance (Ertmer & Newby, 1993) suggests that behaviourism, cognitivism and constructivism are the major perspectives on learning. However, the distinction is relatively small, since it is mentioned that constructivism can be considered as a branch of cognitivism, a perspective which is in line with (Greeno et al., 1996).

Before returning to educational theories in section 2.6 we will take a look at what the Semantic Web is about (section 2.2) and what it offers with respect to supporting knowledge-building discourses (section 2.3-2.5).

### 2.2 Semantic Web Basics

Many of the useful characteristics of Semantic Web follow from the metadata language allowing expressions to be combined effortlessly. Other characteristics are inherited from the Semantic Web being rooted in the regular web. For instance, the unified approach to referencing resources

---

\(^1\) The knowing and motivation/engagement aspects are left out.
via URIs\textsuperscript{2}, allows anyone - not just the owners - to express statements on resources. Furthermore, the vocabularies used to make statements about resources follow the same principles allowing them to be used and extended by others in an evolutionary manner.

At the base of Semantic Web is RDF, \textit{Resource Description Framework} (Manola & Miller, 2004). An RDF expression consists of a set of statements, where a statement is a fact about a resource, for example title, description, creation date, or a relation to another resource. A statement can be seen as a very simple sentence in natural language following the pattern "subject predicate object" where subject and predicate are resources while the object is either a resource or a literal\textsuperscript{3}. In figure 1 we see two statements in graph notation, and in table 1 the same statements are listed in a plain three-column layout.

![Figure 1: RDF graph with two statements, resources are depicted as ellipses and literals as squares.](image)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.com/sheldon">http://example.com/sheldon</a></td>
<td><a href="http://xmlns.com/foaf/0.1/name">http://xmlns.com/foaf/0.1/name</a></td>
<td>&quot;Sheldon Cooper&quot;</td>
</tr>
</tbody>
</table>

Since statements can be represented in a three column table layout they are sometimes referred to as "triples" and repositories containing RDF-graphs as "tripstores."

RDF can be represented in various formats such as RDF/XML (Beckett, 2004) and Turtle (Becket & Berners-Lee, 2011) both of which can be used for exchanging RDF graphs between systems. Below is the RDF graph from above (depicted in figure 2 and listed in table 1) shown in the turtle format:

---

\textsuperscript{2} URI stands for Uniform Resource Identifier which is an Internet Official Protocol Standard for describing global identifiers of resources, see http://www.ietf.org/rfc/rfc3986.txt. The more well-known acronym URL stands for Uniform Resource Locator which corresponds to a special kind of of URI, which - beyond identity - also provides a mechanism for locating and retrieving representations of resources. For example, the URL of a web page (using the http URI scheme) allows the web page to be located and retrieved. In contrast, the URI of a book (using the urn:isbn URI scheme) only provides an identity, which may be used for ordering a copy of the book - but not for directly retrieving a digital representation of it.

\textsuperscript{3} A literal is either a string, a string with a language, or a string with a datatype.
To be able to address a specific set of triples within a larger RDF graph, the concept of a *Named Graph* was introduced in (Carroll, Bizer, Hayes, & Stickler, 2005) and has subsequently been included in other standards such as the *SPARQL Query Language* (Prud’hommeaux & Seaborne, 2008). Named graphs is best understood as a forth column in a table view, consider extending table 1 above, containing identifiers for the Named graph the statements belong to. The Named Graph identifiers are always URIs.

One of the first things you need to do before you can start using RDF to express facts is to decide on a vocabulary to use. The vocabulary is typically a set of URIs to use for predicates, such as in the example above *foaf:name* and *foaf:knows*, but also URIs such as *foaf:Person* that are used to differentiate between different types of resources. To formalize the vocabulary it is suitable to use the *RDF Vocabulary description language*, also referred to as RDF Schema (RDFS) described in (Brickley & Guha, 2004). Another option is to use the *Web Ontology Language* also referred to as OWL, described in (Motik, Parsia, & Patel-Schneider, 2009), that provides a richer set of tools for describing classes, properties, individuals, and data values.

### 2.3 RDF Supports Objective Metadata

The need to express objective metadata, such as title and author of a resource, is a natural part of the publishing process that will not go away. In this process it is common to rely on one or several *metadata specifications* or *metadata standards* that defines terms to be used in the metadata expressions. Two examples of metadata specifications that can be useful when expressing authoritative metadata are *Dublin Core terms* and *IEEE/LOM*. Dublin Core terms are easily expressed in RDF, since its abstract model, the Dublin Core Abstract Model, largely resembles RDF with properties and values used to describe resources. The IEEE/LOM specification, on the other hand, has an abstract model that closely resembles the XML infoset with element-in-element that are harder to translate to RDF. Despite this, in (Nilsson, Palmér, & Brase, 2003) an RDF binding of the IEEE/LOM specification has successfully been introduced. The RDF expression of IEEE/LOM has later been taken up within the Joint DCMI/IEEE LTSC Taskforce. Although no final specification has been produced, there is a draft mapping between IEEE/LOM and the Dublin Core Abstract Model which indirectly gives an RDF expression.

It could be argued that this lack of a definitive expression of IEEE/LOM in RDF is a hindrance for using RDF in the learning domain. However, there are other factors to take into account. For example, in his thesis *From Interoperability to Harmonization in Metadata Standardization*. (Nilsson, 2010) makes a detailed analysis of the problems regarding metadata interoperability across metadata standards and points to several problems with syntactical approaches based on

---

7. [http://www.w3.org/TR/xml-infoset/](http://www.w3.org/TR/xml-infoset/)
8. [http://dublincore.org/educationwiki/DCMIIEEELTSCTaskforce](http://dublincore.org/educationwiki/DCMIIEEELTSCTaskforce)
XML. Nilsson argues that a common abstract model and an accompanying schema model should be used when expressing metadata standards and points out that the only realistic candidate as of today is RDF and RDF Schema.

Given that relevant metadata standards have reasonably stable expressions in RDF, they have the potential to support objective metadata. However, there is also the important question of authority (which can be assumed to imply truthfulness). This concerns who stands behind the metadata and what credibility they have for their issued statements about the resource in question. The fact that a specific metadata standard is used to state more or less objective facts about resources does not make a specific metadata record authoritative. But, if the metadata record can be retrieved from the same origin (as indicated by their URIs) as the resource it describes it should probably be considered authoritative. However, in general this is not the case and one has to rely on trusting specific content/metadata providers instead.

2.4 RDF Supports Subjective Metadata

As described in paper 2, one of the most severe misconceptions about metadata, is the idea that metadata only should be used to express objective information. In contrast, metadata can be used to express subjective information, such as comments, opinions, tags, ratings, reviews etc. Even though established metadata standards like IEEE/LOM can be used for this purpose, it is a clumsy tool for the job. Especially if you consider the scenario that subjective information is expressed over a prolonged period by different users, and perhaps also across different systems. In such cases it is not realistic to repeatedly collect the subjective information into a single authoritative metadata document (which is the predominant assumption of how these standards are to be used. See paper 2 for a longer discussion of the document centric view on metadata).

Consider the example of tagging. A single resource may be tagged by many different users. The reasons for providing each tag may differ, and even if the tag is the same for different users, there is no guarantee that the intended meaning is the same. Hence, just collecting the tags into a single authoritative metadata document makes no sense. Still, it would be valuable if the tags related to a resource could be connected and the intentions behind each tag could be made clear. One initiative that aims for this is the MOAT project⁹ (Meaning Of A Tag), described in (Passant & Laublet, 2008). MOAT is a good example of how RDF technology can be used for supporting the expression of subjective metadata.

An older project which is capable of expressing subjective metadata using RDF technology is Annotea, which is described in (Koivunen, 2005). Annotea provides a solution for annotating web documents, or even parts of web documents. Simply put, the client, which is typically a browser plugin, connects to an Annotea server from where relevant annotations are retrieved and displayed on top of the currently shown web document. New annotations, private or shared, can be pushed to the Annotea server by the client. Note that the annotations can represent different things, in (Koivunen, 2005) annotations, bookmarks, and topics are mentioned.

Both MOAT and Annotea are nice initiatives, but they only solve rather specific problems regarding subjective metadata. A more generic approach for taking care of subjective metadata is provided by EntryScape/EntryStore which is discussed in section 4.1 and chapter 6.

⁹ http://moat-project.org
2.5 RDF Supports an Evolving Human Discourse

Let us list three fundamental and important characteristics of RDF that are relevant when used for supporting an evolving human discourse:

**Talk about anything** - statements can be made about any resource, that is anything that has been given an identifier, often in the form of a URL. Examples include not only web pages but also physical objects like cars, people, and books in a library as well as ideas or abstract objects like the concept of a perfect circle.

**Unlimited set of terms** - there are no conceptual bounds on what can be expressed, if there are no established terms that match your need, you can simply invent them yourself on the fly. Of course, there is no guarantee that such new terms will be understood directly by others, but this is similar to all other languages, including spoken languages like English.

**Reuse terms in new contexts** - you can combine any terms you like, if someone introduced or defined a term and you find it useful, you can use it. But if you use it in a way that is not coherent with the original intention, the result may be confusing - just like any other misuse of language.

One difference with respect to natural languages is that even though RDF is powerful in it's expressiveness, it always consists of sets of simple statements, that is, no complex sentences to parse. Another difference is that the use of globally unique identifiers, in the form of Uniform Resource Identifiers (abbreviated as URI), avoids unnecessary ambiguities when referring to resources. Unfortunately this also makes RDF less readable for humans, but as RDF should not be exposed to end users directly this is not a real problem.

2.6 The Semantic Web and Design Principles for Learning Applications.

Many have argued the case for using Semantic Web technologies for learning. For example both (Anderson, 2008) and (Ohler, 2008) claim that the added value originates from the use of intelligent agents that fetch and aggregate information. This is largely in line with the original vision of Tim Berners-Lee, see (Berners-Lee, Hendler, & Lassila, 2001). However, there are other values for learning that can be provided by the Semantic Web.

We now continue the discussion started in 2.1 and examine how the three learning perspectives presented there have been broken down by (Greeno et al., 1996) into six design principles for learning applications. For each principle we discuss which added values Semantic Web technology could provide in relation to presently prevailing (non-semantic) technologies. Moreover, an attempt is made to quantify the added-value into three levels (minor/moderate/major).

The design principles listed below are named (b1)-(b3) for the behaviourist/empirist perspective, (c1) for the cognitivist/rationalist perspective and (s1)-(s2) for the situative/pragmatist-sociohistoric perspective.

**Design principle (b1) - Routines of activity for effective transmission of knowledge**

---

10 The design principles are actually stated for learning environments but this includes learning applications.

11 The naming conventions are taken from (Greeno, Collins, & Resnick, 1996).
"Learning activities can be organized to optimize acquisition of information and routine skill. In learning environments organized for these purposes, learning occurs most effectively if the teaching or learning program is well organized, with routines for classroom activity that students know and follow efficiently."

(Greeno et al., 1996) p. 27

Learning Management Systems (LMS), such as Moodle and Blackboard, excel at supporting well (= strongly) organized instruction, often with a focus on the needs of the teacher. They provide an authoritative source for getting hold of learning material, testing your knowledge, and providing overall guidance of the expected learning activities. Many standards have been developed to support interoperability between LMS:es. The most known is probably the SCORM standard\(^\text{12}\) introduced by ADL\(^\text{13}\) as a collection of technical standards for describing, sequencing, packaging and executing learning material. It is feasible to improve upon these standards by expressing them in RDF. For example, an early attempt for describing learning object in RDF was made by (Nilsson et al., 2003). Such RDF-based expressions of learning objects (and maybe also for tests, sequencing of learning objects etc.) definitely have the potential for broader reuse within more general contexts, which do not necessarily involve learning. However, it is not realistic to think that, within the near future, major LMS:es would switch from SCORM to another standard expression. This is due to two reasons: First, achieving compliance with SCORM in existing LMS:es has taken a long time (and is still 'patchy'). Second, to this date, the achieved reuse due to SCORM compliance has not been impressive (Gonzalez-Barbome & Llamas-Nistal, 2007).

From this we conclude that incremental improvements of existing learning technology standards are probably not a sufficient motivation to trigger an uptake of Semantic Web technologies in major LMS:es. Although the pressure to integrate with web content in general may push LMS:es to better support Semantic Web or similar technologies (such as for instance RDFa\(^\text{14}\) and Schema.org\(^\text{15}\)).

Hence, we conclude that the added value of applying Semantic Web for design principle (b1) is minor, at least as long as the main supportive technology for these kind of activity-routines are LMS:es.

**Design principle (b2) - Clear goals, feedback and reinforcement**

For routine learning, it is advantageous to have explicit instructional goals, to present instructions that specify the procedure and information to be learned and the way that learning materials are organized, to ensure that students have learned prerequisites for each new component, to provide opportunities for students to respond correctly, to give detailed feedback to inform students which items they have learned and which they still need to work on, and to provide reinforcement for learning that satisfies students' motivations.

(Greeno et al., 1996) p. 27

From a technological perspective, the distinction between design principle (b2) and (b3) (presented in the next paragraph) is too small to motivate a separate discussion and therefore both design principles will be discussed together below.

---


\(^\text{13}\) Advanced Distributed Learning is an initiative originating in the Unites States Department of Defence, http://www.adlnet.org/

\(^\text{14}\) RDFa provides a way to embed RDF triples into HTML.

\(^\text{15}\) Schema.org is an initiative to allow simple markup of web pages in a way recognized by major search providers.
Design principle (b3) - Individualization with technologies

Acquisition of basic information and routine skills can be facilitated by using technologies, including computer technology, that support individualized training and practice sequences.

(Greeno et al., 1996) p. 27

An Intelligent Tutoring System (ITS) is a kind of expert system that tries to combine knowledge of the student (the learner model), knowledge of the topic (domain model) and good teaching practices (pedagogical model) to provide individualized instruction. ITS systems appeared in the middle of the seventies and interest peaked in the late eighties for two major reasons (Reeves, 1998): First, a lack of impact on mainstream education and, second, technical difficulties inherent in building student models and facilitating human-like communications, difficulties which had been greatly underestimated by proponents of this approach.

In addition to Intelligent Tutoring Systems, there are also Adaptive Educational Hypermedia Systems (AEHS) that, based on information collected around an individual, try to provide a personalized experience by individually adapting the available navigation paths through some material. In the nineties, both ITS:es and AEHS:es changed their focus in order to accommodate web based systems, in recognition of the growing importance of the web. According to (Brusilovsky & Peylo, 2011), when ITS:es and AEHS:es were refocused as web-based systems, they were found to have a large overlap. Consequently, a new area called Adaptive and Intelligent Web-based Educational Systems (AIWBES) was introduced in order to provide a more systematic view on the techniques of ITS, AEHS, as well as on a range of other related techniques.

Although not always using this term AIWBES, many have argued for the use of Semantic Web to improve upon AIWBES techniques, such as (Aroyo & Dicheva, 2004), (Henze, Dolog, & Nejdl, 2004), (Devedzic, 2004), and (Devedžić, 2006). Within the area of ITS, ontologies can be used to capture various learner-, domain- and pedagogic models. These ontologies, and data expressed with the help of them, can be shared across systems, which provides added value when problem domains overlap and become increasingly complex. Within the area of AEHS (and ITS as well) using ontologies may help to avoid the so called "cold start problem" by reusing learner models across systems. In addition, interoperability aspects of Semantic Web technologies make more learning material available for AIWBES systems, a fact which should allow for better adaption to learners needs.

However, despite the fact that AIWBES systems have been around for a long time, there seems to be little or at most moderate uptake. Hence, this thesis argues that there is only moderate added value of Semantic Web with respect to design principles (h2) and (b3).

Design principle (c1) - Interactive environments for construction of understanding

Learning environments can be organized to foster students' constructing understanding of concepts and principles through problem solving and reasoning in activities that engage students' interests and use of their initial understandings and their general reasoning and problem-solving abilities.

(Greeno et al., 1996) p. 27

---

When a learner enters a new system the learner model is empty and the learner needs to either provide information in some way, for instance by taking a test, or accept initially week personalization.
From a technological perspective, the distinction between design principle (c1) and (s1) (presented in the next paragraph) is too small to motivate a separate discussion and therefore both design principles will be discussed together below.

**Design principle (s1) - Environments of participation in social practices of inquiry and learning**

Learning environments can be organized to foster students' learning to participate in practices of inquiry and learning and to support the development of students' personal identities as capable and confident learners and knowers. These activities include formulating and evaluating questions, problems, conjectures, arguments, explanations, and so forth, as aspects of the social practices of sense-making and learning, including abilities to use a rich variety of social and material resources for learning and to contribute to socially organized learning activities, as well as to engage in concentrated individual efforts.

(Greeno et al., 1996) p. 27

According to (Jonassen, 1999) a Constructivist Learning Environments (CLE) is a system where a problem\(^{17}\) drives the learning - rather than a topic. A CLE should provide a context for the problem, as well as a representation or simulation of it that should be both appealing and authentic\(^{18}\). In addition, there should also be a way for students to actively work and familiarize themselves with the problem in a practical manner. (Jonassen, 1999) continues by looking into three kinds of tools\(^{19}\) that are useful in a CLE. Students need to find information to gain deeper understanding of the problem, and they can do so by using information-gathering tools that predominantly work with the web. Moreover, students need to be able to actively work, develop and construct solutions to the problems, and therefore they need knowledge construction tools such as for example visualization and knowledge modeling tools. Finally, in order to support teamwork among students, there is a need for conversation and collaboration tools.

A natural question to ask is whether CLEs can be turned into unified environments in analogy with how LMS:es are constructed today\(^{20}\). One strong argument against the idea of a CLE as a unified environment is that the tools mentioned above are diverse and hard to integrate. The alternative, to replace them with smaller, dedicated tools that fit into such a unified environment would require a lot of effort. The resulting tools would most probably not appear authentic, which is an important requirement of constructive environments. Although, with the arrival of Web2.0, it has become more feasible to aim for some form of unified CLE, since tools are increasingly available as online services, a fact which sometimes allows them to be embedded into other environments.

Semantic Web can provide added-value for CLEs in several ways. If Semantic Web technologies were used more broadly in markup of web-pages, it would substantially improve the effectiveness of information-gathering tools. For instance, the Schema.org initiative mentioned above helps search engines to improve search results based on Semantic Web information\(^{21}\). Moreover, if knowledge creation tools were based on Semantic Web technologies, the integration with CLEs would become easier, since there would be less need to develop support for new APIs.

---

17 For simplicity the word ‘problem’ is used in the text, it may also be a ‘question’, an ‘issue’, a ‘case’, or a ‘project’.
18 Authentic in the sense that it is realistic and relevant for the current context and hopefully also in the future.
19 Note that they represent aspects that are not mutually exclusive.
20 Where tools are developed specifically for each LMS.
21 For example, Googles knowledge graph in part originate from Schema.org based markup, the effect is (at this point in time) improved searches with disambiguation and fact sheets for recognized entities.
for every tool. One example of a knowledge creation tool is provided by Conzilla (see section 4.2), which uses RDF to produce new knowledge as well as to relate to existing knowledge. This use of RDF makes Conzilla compatible with a larger ecosystem of Semantic-Web-enabled knowledge construction tools. It is important to emphasize that Semantic Web technologies can be instrumental in handling the interoperability problems between various web2.0 systems, see (Bojars, Breslin, Peristeras, Tummarello, & Decker, 2008). The approach, termed Social Semantic Web, has been discussed further by several others such as (Jeremic, Jovanovic, & Gasevic, 2011) and (Breslin, Passant, & Decker, 2009).

In summary, there is a real possibility that Semantic Web will provide a major added value to design principles (c1) and (s1), especially since the set of Web2.0 technologies is still growing.

### Design principle (s2) - Support for development of positive epistemic identities

*Learning environments can be organized to support the development of students' personal identities as capable and confident learners and knowers. This can include organizing learning activities in ways that complement and reinforce differences in patterns of social interaction and in expertise brought by students of differing cultural backgrounds.*

(Greeno et al., 1996) p.27-28

Today, online identity for many people is fragmented across different social networks and (social) web sites. This fragmentation may be intentional since it serves a way to reveal different facets of a user's personality. However, it can also be harmful since people often need to spend a lot of time in forming their relationships and building their reputation. This problem has been discussed in more detail, for example in a chapter on trust and privacy in (Breslin, Passant, & Vrandečić, 2011). Here it is also argued that Semantic Web may remedy the situation by allowing better control of which identities to keep separate and which to merge.

Social networks of today allow the forming of identities by establishing relations to others as well as exposing activities in the form of positive feedback, status updates, or longer texts for others to see. Social Networks with a more professional touch, such as LinkedIn, also allow people to provide a resumé. However, to support the collection of resources that you care about in general - be it links or documents that you have produced - current social networks in general have little to offer. E-portfolios may be useful here, and the example of Confolio/EntryScape (see section 4.1), shows that Semantic Web technology can be useful by allowing a wide range of expressions.

In summary, Semantic Web technologies have major potential for providing added value for this design principle (s2). This is mostly due to the large use of social networks and the capability of Semantic Web technologies to strengthen the identities of participators in learning situations.

Having thus discussed all of the design principles, we now summarize our findings in table 2, including the estimated potential added values and the corresponding aspects of RDF that these values depend on.

---

22 Such as for example Protégé and IsaViz, unfortunately not many other such tools exists at this point in time.
23 Positive feedback take different forms on different social networks, for example Facebook's "like" and Google's "+1".
Table 2: Design principles for learning environments together with potential added value from Semantic Web and which aspects of RDF used. OM stands for Objective Metadata, SM for Subjective Metadata and EHD for Evolving Human Discourse, see sections 2.3, 2.4, and 2.5 respectively.

<table>
<thead>
<tr>
<th>Design Principles</th>
<th>Added value of Semantic Web</th>
<th>Based on</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>Minor</td>
<td>OM</td>
</tr>
<tr>
<td>b2-b3</td>
<td>Moderate</td>
<td>OM</td>
</tr>
<tr>
<td>c1-s1</td>
<td>Major</td>
<td>OM, SM, EHD</td>
</tr>
<tr>
<td>s2</td>
<td>Major</td>
<td>OM, SM</td>
</tr>
</tbody>
</table>

From the above discussion it is clear that there is substantial benefits in using Semantic Web technologies for learning, especially from the cognitivist/rationalist (c1) and situative/pragmatist-sociohistoric perspectives (s1)-(s2). In addition it indicates that subjective metadata and evolving human discourse are more interesting aspects of RDF than supporting the expression of objective metadata, although the latter is a necessary foundation for the former.

This conclusion is also in line with the vision stated in the introduction of this thesis:

Learning applications based on Semantic Web technologies will allow people to express themselves and communicate with increased precision on a growing and changing range of topics.

Although expressed differently, the overlap of this vision with cognitive/rationalist and situative/pragmatist perspectives are quite clear. Moreover, the vision is partly based on the Knowledge Manifold architecture, introduced in (Naeve, 2001a), that outlines a somewhat different approach to learning in a networked environment. It emphasizes the creation of knowledge as an interconnected conceptual 'patchwork' constructed by different "knowledge gardeners" that maintain their individual "knowledge patch". Learners can find their way through this "knowledge patchwork" by conceptual browsing, using tools such as introduced in (Naeve, 2001b). In this way, the knowledge manifold architecture can be used to build communities of learning/practice where the participators can communicate and build upon each others' knowledge in many different ways.

The idea of an Knowledge Manifold architecture has been used as an inspiration for how the Semantic Web can be used for learning, leading to the idea of a Human Semantic Web (Naeve, 2005), which was discussed already in paper 2. More broadly, chapter 2 of paper 2 argues that a Semantic Web-based architecture can allow metadata to be subjective and non-authoritative, evolving, extensible, distributed, flexible and conceptual.

However, this thesis claims that in the end, no matter which perspective one applies on learning, the new digital possibilities all carry with them an increased need to support more complex forms of human communication and interaction. In this context, Semantic Web arguably has an important role to play.
2.7 The Implicit Requirements of the Semantic Web

It is interesting to consider what would happen if the starting point of this thesis was not Semantic Web technology based on RDF but rather the aspects of RDF described in this chapter reformulated as requirements. Any approach to express information in the learning domain would at least need a way to state facts around things that could be identified globally in a unique manner. It would also need mechanisms to define new vocabularies and extend existing ones. To make sense of such vocabularies, their meaning would have to be understood by many parties, and a practice to evolve the corresponding discourse by introducing new vocabularies would be highly beneficial.

It could be argued that such a solution would share many of the traits of the Semantic Web of today. Moreover, from a pragmatic perspective there is much to gain from building upon what has already been established, since there are already tools to use, existing knowledge among developers etc. Furthermore, Semantic Web technology is defined within the W3C that drives the laborious process of reaching community consensus around recommendations. This indicates that you need really good reasons for trying to establish new technologies that largely overlap with already existing W3C recommendations, since these recommendations have been discussed and modified back and forth among stakeholders to fit a range of different needs and requirements.

2.8 Summary

This chapter has provided a brief presentation of three major perspectives on educational theories, as well as a brief introduction to Semantic Web technologies, more specifically to RDF. We have seen how important aspects of RDF can be used to support (i) objective metadata, (ii) subjective metadata, and (iii) an evolving human discourse. From the perspective of the presented educational theories, the aspects (ii) and (iii) have been concluded to be the most important with respect to the derived added value of Semantic Web technologies for building learning applications.

Related to the three perspectives on educational theories, six different design principles for learning applications have been discussed, with a focus on the added values of using Semantic Web technologies. It has been shown that there is some added value within each of these principles, although the largest benefits have been identified for design principles derived from the cognitivist/rationalist- and the situative/pragmatist-sociohistoric perspective.

Finally, this chapter has discussed what would happen if the starting point of this thesis was not the current Semantic Web, but rather the need to express facts around things that have globally unique identifiers, as well as a few other related requirements regarding the process of incrementally defining terms. The conclusion drawn was that the resulting technology would probably be similar. Moreover, the importance of the consensus building process within W3C that had led to the current design of Semantic Web technologies should not be underestimated.

This thesis argues that it is (i) the useful characteristics of RDF, (ii) the potential added value of Semantic Web from a learning perspective, and (iii) the pragmatism of relying on something that is already widely known that makes Semantic Web a good basis for building learning applications.

24 W3C stands for World Wide Consortium, more information can be found at http://w3.org/.
3. Architectures, Technologies and Application Types

The previous section has provided theoretical arguments for how Semantic Web technologies can be useful for learning applications. This section will provide background knowledge of architectures and technologies as well as a new categorization of applications into types that will come in handy when reading this thesis and applying its results.

In addition, this chapter will also provide some insight in how the various technologies and architectures relate to each other as well as to the application types. Since Semantic Web technology is based upon the Web Architecture, which will be described in section 3.2, we will use the Web Architecture as the point of departure. It will also be used as a "simplest common grounds" when comparing how architectures and technologies relate.

3.1 Web Architecture

The Architecture of the World Wide Web, (Jacobs & Walsh, 2004) is a W3C recommendation since December 2004. According to this document the architecture of the web can be divided into three bases:

**Identification** - conceptual resources are given global identifiers according to the URI specification. Many resources, such as web pages and images are information resources, that is, they have representations that can be sent as messages.

**Interaction** - communication between agents over a network involves URIs, messages and data. The communication is facilitated via a range of web protocols.

**Formats** - provide agreement of how to express representations of resource states. Such an agreement includes both a syntax to encode data and a semantics prescribing how it should be used. The web architecture provides no restrictions on which formats to use, although reuse is encouraged in order to increase interoperability.
In figure 2 these three bases of the Web Architecture are depicted in the background, with other architectures and technologies on top. When architectures or technologies have common ground they are drawn so that they overlap. Even though the figure is quite detailed there are areas where it falls short, although hopefully the overall message of how the architectures and technologies overlap should be reasonably clear. The figure, especially all the abbreviations, will not be explained here, but is intended only as orientation material. The reader is encouraged to return to this figure when reading about specific architectures and technologies in the following subsections.

The web architecture document proceeds to outline 5 principles to capture the fundamental properties of the web, 3 constraints that are consequences of the design choices made, and 24 best practices that, if followed, are believed to increase the value of the web. These principles and constraints carry more weight and are especially important to consider when new web technologies are developed. The W3C utilizes a community consensus process where the standards it develops are always reviewed by the member organizations, before being approved. Consequently, the standards are anchored among the practitioners and technology providers.

---

25 At the time of writing there are 345 members organizations at W3C.
when released, and therefore they often have a high impact. The following statement (taken from section 1.1 of the recommendation) further elevates the importance of the Web Architecture document in a broader sense:

*This document describes the properties we desire of the Web and the design choices that have been made to achieve them. It promotes the reuse of existing standards when suitable, and gives guidance on how to innovate in a manner consistent with Web architecture.*

For the purpose of this thesis the Web architecture is important both as a precursor for the semantic web (see section 3.2) and as a good choice for building applications for learning (see section 3.7).

In the following sections we will discuss how various technologies and architectures relate to the Web Architecture. The principles, constraints and best practices of the Web architecture are introduced when needed for the discussion. For a complete listing the reader is referred to the Web Architecture recommendation (Jacobs & Walsh, 2004).

To simplify the discussion below, the terms *utilize* and *integrate* are introduced with respect to Web architecture. That a technology or architecture *utilizes* the Web Architecture means that it relies on the three bases, that is, identifiers, interactions, and formats as prescribed in the Web Architecture recommendation. That a technology or architecture *integrates* with the Web architecture means that it follows the best practices: *identify with URIs, link identification, web linking,* and *generic URIs*. To put it more plainly, integration means that resources should be identifiable by URIs and that the formats used should allow expression of links, or relationships, by using those URIs.

It is argued in this thesis that technologies and architectures that both utilize and integrate with Web Architecture are better equipped to strengthen and complement each other than those that only utilize or have no relation to Web Architecture at all. In the last section of this chapter, section 3.8, all the relations to the Web Architecture are summarized and visualized in a diagram.

### 3.2 Semantic Web Data

Let us now take a brief look at how Semantic web data, that is, RDF, relates to the Web Architecture. For those readers who need a short introduction to RDF section 2.2 should be enough; for a longer treatment see the RDF primer (Manola & Miller, 2004). The following generic statement introduces the W3C recommendation on RDF concept and abstract syntax (Klyne & Carroll, 2004):

*The Resource Description Framework (RDF) is a framework for representing information in the Web.*

RDF represents information by making statements on resources via their identifiers, that is, their URIs. The statements can be joined together into larger graphs allowing more complicated expressions to form. This is possible due to the fact that there are well defined rules for joining two graphs without changing the semantics. A consequence is that the graph can be distributed and ownership of parts of the graph can be handled by the same principles as the web itself, that is, via the IANA URI scheme registry and the DNS. There are several ways to locate RDF graphs: they can be embedded in various other forms of markup formats, accessed directly via
3. Architectures, Technologies and Application Types

HTTP or searched via the SPARQL protocol for querying large RDF datasets. There are several interchangeable formats for transporting RDF graphs, the most widely known is the RDF/XML format, which is a W3C recommendation, see (Beckett, 2004). We can now conclude that:

1. RDF utilizes all three bases, identifiers, interactions and formats of the Web Architecture.
2. RDF integrates with the Web Architecture, since RDF data, the RDF statements, makes use of URIs to express relationships between resources.

Another strength of RDF is that statements can be made about resources which have no digital representation, for instance physical objects. This seems to break the Web architecture's best practice of available representation. However, the W3C note on Cool URIs for the Semantic Web, see (Sauermann & Cyganiak, 2008), provides guidelines of how to circumvent this problem by using either hash URLs\(^\text{26}\) or the HTTP status code 303 see other.

3.3 Service-Oriented Architecture

Web Services have proven to be a powerful tool for defining interfaces to various systems. However, their use together with Semantic Web technology is less impressive. One reason may be that RDF semantics has a declarative character (stating facts on resources) while Web Service semantics has a procedural character (invoking methods at a distance). More specifically:

The semantics of RDF is about how to interpret statements on resources. This is quite different from the semantics of a Web Service which is bound to the actions it performs when invoked. This means that Web Services in general have no understanding of resources, nor for that matter of statements about resources. Clearly, it is possible to define specific Web Services that have knowledge of resources and statements of resources, but it is not prescribed by the Web Service standards.

Let us dig deeper by taking a look at the W3C note on Web Services Architecture, see (Booth et al., 2004), where a web services is defined as:

> a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Web Services provides a standard way of interoperating between different software applications, running on a variety of platforms and/or frameworks. The software applications that use a web service should do so in accordance with a shared meaning, semantics, of what the real world effects of using it are. Hence, the semantics of web services are about the actions that are described using the Web Service Description Language, WSDL. Consequently, a SOAP-message - a piece of data, transported via a web service - is used to identify and provide necessary information to allow the action to be invoked via the web service.

The Web Service architecture utilizes the Web Architecture in the sense that it relies on identifiers to identify web services, web protocols to carry the interactions, and formats to encode the actions as data. However, it is questionable whether the Web Architecture principle of safe retrieval is fulfilled. Safe retrieval means that it is possible to retrieve resource

\(^{26}\) Hash URLs are URLs that have a part of the address after a hash ("#") character.
representations in a manner which has no side-effects. The latest version of the SOAP standard includes the *request message exchange pattern* that supposedly fixes this gap. The idea is that implementors should distinguish resource retrieval from other RPCs (Remote Procedure Call), map them to URIs and support plain HTTP GET to retrieve them instead of HTTP POST with a SOAP header. Unfortunately, since this approach is not mandatory, few implementors provide support for this feature (see for instance section 3.2 in (Newcomer, Laskey, & Hégaret, 2007) where this is discussed). Hence, in practice Web Services only partially *utilize* the Web Architecture.

Furthermore, there is no requirement that the exchanged data, that is the SOAP-messages, use URIs to link to other resources. Hence, Web Services do not *integrate* well with the Web Architecture either.

### 3.4 REST

The REST (REpresentational State Transfer) architectural style was introduced by Roy Fielding (Fielding, 2000) and is widely claimed to be the architectural style of the Web. Although, somewhat surprisingly, this does not mean that all of its constraints have been adopted by the Web Architecture as described in the W3C recommendation (Jacobs & Walsh, 2004). According to Fielding, an architectural style means a set of architectural constraints. REST consists of 6 architectural constraints, of which one is the uniform interface which has four additional interface constraints. In table 3 the constraints are listed and shortly explained. If matching principles, constraints, and practices of the Web architecture exists, they are listed in the third column. Note that since REST is an architectural style and not an actual architecture or technology we cannot really claim that REST *utilizes* or *integrates* with the Web Architecture. However, table 3 and the following discussion aims to shine some light on the similarities and differences with respect to the constraints.

*Table 3: REST constraints explained and compared with the Web Architecture.*

<table>
<thead>
<tr>
<th>REST Constraint</th>
<th>Explanation</th>
<th>Web architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-Server</td>
<td>Clients initiate communication with a server over a network.</td>
<td>-</td>
</tr>
<tr>
<td>Layered system</td>
<td>Allows services and complexity to be hidden behind interfaces.</td>
<td>-</td>
</tr>
<tr>
<td>Stateless</td>
<td>Every request must contain all information needed to understand it.</td>
<td>-</td>
</tr>
<tr>
<td>Code-on-demand</td>
<td>Allows richer clients without changing the underlying system.</td>
<td>-</td>
</tr>
<tr>
<td>Cache</td>
<td>Responses should include caching constraints to improve network efficiency.</td>
<td>* Safe retrieval</td>
</tr>
</tbody>
</table>
### Uniform interface

Standardized interfaces improve simplicity, visibility of interactions and decoupling of implementations from the services they provide. The uniform interface is described further by the following four sub-constraints:

- **Identification of resources**
  - Resource identifiers are used to consistently identify a resource over time, for example in interactions between components. The identifier of a resource may be used to gain access to zero or more resource representations.
  - *Global identifiers*  
  - *Identify with URIs*  
  - *URIs Identify a Single Resource*  
  - *Avoid URI aliases*  
  - *Consistent URI usage*  
  - URI opacity

- **Manipulation of resources through representations**
  - Components perform actions on a resource by using a representation to capture the current or intended state of that resource and by transferring that representation between components.
  - *Reuse representation formats*  
  - *Available representation*  
  - *Consistent representation*  

- **Self-descriptive messages**
  - In addition to representation data a message should contain control data that defines the purpose of a message between components. It may also contain metadata for both the representation and the resource, independent of the representation.
  - *Metadata association*

- **Hypermedia as the engine of application state**
  - An application, viewed in a browser, moves from one state to the next by someone examining and choosing from alternative state transitions (links) in the current set of representations.
  - *Link identification*  
  - *Web linking*  
  - *Generic URIs*  
  - *Hypertext links*

From this overview, let us focus on those REST constraints that are not covered in the Web Architecture. First, the Web Architecture cannot restrict to client server interactions without breaking compatibility with Web Services, XMPP\(^{27}\), peer-to-peer networks etc. Second, in the context of networked based systems the layered system constraint is limited to the combination with the client-server constraint according to Fielding (see section 3.4.2 in (Fielding, 2000)). Hence, there is no reason to reflect the layered system constraint in the Web Architecture either. Third, a large portion of the web is today driven by software that keeps application state in sessions on the server-side which directly contradicts the stateless REST constraints. Consequently, the Web Architecture cannot adopt this requirement without excluding a large part of the current web. Fourth, the code-on-demand constraint is actually an optional constraint of REST and is not covered explicitly in the Web Architecture. However, there is a best practice that prescribes *separation of content, presentation, and interaction* for data formats that taken

\(^{27}\) Extensible Messaging and Presence Protocol, see http://xmpp.org
together with a section on extensibility points in the direction of code-on-demand. Finally, it is worth noticing that most of the uniform interface constraints are covered by best practices rather than the more heavyweight principles.

From this comparison it is perhaps not surprising that the Web Architecture recommendation does not state a relation to, or even mentions REST. Going through revisions of the document shows that REST was originally included but was removed late in 2002\(^\text{28}\). A likely explanation is that the rise of, among other things, the Web Services standards resulted in a need to accommodate them as part of the Web Architecture. And since Web Services does not fulfill the architectural constraints of REST, the Web Architecture - which seems to be a kind of umbrella for the work done in W3C - could only include parts of the architectural constraints of REST. In fact, the only compulsory constraints that overlap between REST and the Web Architecture are the use of global identifiers and safe retrieval. However, even though the Web Architecture does not enforce REST, it certainly accommodates REST.

### 3.5 Resource Oriented Architecture

The ROA (Resource Oriented Architecture) was introduced in 2007 (Richardson & Ruby, 2007) for building Web Services that are in line with the REST principles. An important reason for introducing ROA was that REST only provides a set of architectural constraints and not an architecture in itself. Practitioners have been arguing over the correct approach to realize the constraints of REST, leading to heated debates as well as to slightly different approaches. With ROA, Richardson and Ruby collected best practices into a consistent and concrete architecture that covers common needs when building Web Services according to REST constraints. ROA tries to follow the constraints of REST, although presented in a slightly different way.

...I introduce the moving parts of the Resource-Oriented Architecture: resources (of course), their names, their representations, and the links between them. I explain and promote the properties of the ROA: addressability, statelessness, connectedness, and the uniform interface. I show how the web technologies (HTTP, URIs, and XML) implement the moving parts to make the properties possible.

In table 4, an approximate mapping is provided between the parts and properties of ROA with the constraints of REST. In addition, ROA is constrained to the use of HTTP, URIs and XML (together with other capable formats). These additional choices of technology matches the two remaining REST constraints that are not explicitly mentioned in the table, that is, the client-server and the layered system constraints.

<table>
<thead>
<tr>
<th>ROA parts &amp; properties</th>
<th>REST constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td>Identification of resources</td>
</tr>
<tr>
<td>Resource names</td>
<td>Identification of resources</td>
</tr>
<tr>
<td>Links between resources</td>
<td>Identification of resources</td>
</tr>
</tbody>
</table>

\(^{28}\) REST still remains in the reference list though, certainly an oversight.
Again, it should be noted that ROA is not an attempt to describe the whole web, rather - just as the title "RESTful Web Services" says - it focuses on the programmable web.

In the terminology introduced in section 3.1, it is clear that ROA both utilizes the web's resources, interactions, and formats and integrates well due to its focus on links between resources as well as its connectedness requirement. Richardson and Ruby proceed further by providing a generic process for designing a service. This process provides guidance for how to split the problem space into RESTful resources and decide which interaction the resources should support.

3.6 Linked Data

By combining the hyperlinked character of the Web Architecture with RDF, we get the Linked Data initiative, see (Berners-Lee, 2006) and (Bizer, Heath, & Berners-Lee, 2009). Linked Data is defined by the following rules:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs so that they can discover more things.

Since the introduction of the Linked Data initiative in 2006, many datasets have been made available according to these rules. And since they link to each other, the result is a big interconnected web of data which is growing every day. Note that Linked Data is sometimes referred to as the web of data. Clearly, Linked Data both utilizes and integrates with the Web Architecture.

Linked Data has remained a web of read-only data, but a range of initiatives have appeared lately that suggest mechanisms to write data back. Tim Berner-Lee wrote a design note in 2009 on how he conceives that Linked Data could be made writable (Berners-Lee, 2009). He suggests two approaches, first outlining how documents containing RDF can be written back using HTTP PUT and second how to change individual triples by sending update requests to the server via SPARQL Update messages. The document approach has since been included in the SPARQL 1.1 Graph Store HTTP Protocol, see (Ogbuji, 2012). The SPARQL 1.1 update, see (Gearon, Passant, & Polleres, 2012), is a language that is supposed to be used in an RPC-like fashion via the
SPARQL 1.1 Protocol for RDF, see (Feigenbaum, Williams, Clark, & Torres, 2012). SPARQL 1.1 Graph Store HTTP Protocol on the other hand takes the RESTful approach and outlines how to retrieve and modify named graphs using HTTP methods directly.

It could be argued that Linked Data is a natural consequence of focusing on HTTP and URIs and properly following the REST constraints (especially the hypermedia as the engine of application state constraint, see table 3 in section 3.4 above, which is somewhat overlooked in many RESTful implementations of today).

The most common format used for exposing Linked Data is RDF/XML. However there are other alternatives. One alternative is to use RDFa, see (Adida, Herman, Sporny, & Birbeck, 2012), a technology to enrich HTML with semantics which can be extracted and turned into an RDF graph. However, this pinpoints a difficult point regarding how to design the Linked Data layer. If the starting point is to expose Linked Data as a Web Service for various applications to consume and one of the formats one can get is RDFa-enriched HTML, then all is well. Developers can 'browse' the Linked Data RESTful web service and choose cleaner data formats when they start to develop, if they so prefer.

However, if the starting point is a range of web pages forming a web application which is enriched with RDFa, then other applications, with a different focus, may have problems with accessing the needed information in a suitable manner. Furthermore, if the web applications evolve, as they often do, due to change of user needs, new functions, improved design etc., then the data expression changes as well which might break other applications. RDFa can still be used for structuring data within an application without considering the added value of interoperability and reuse in new contexts. Without considering these added values it can be argued that simpler in-page data-structures such as semantic HTML, XML or JSON can serve the application's needs just as good as RDFa. This would weaken the case for Linked Data.

3.7 Applications Types

This section will look into a range of different application types that are relevant in the learning domain.

A general observation is that a low threshold for starting to use an application increases its chances for reaching its target audience, at least when there is no mandate or possibility to enforce usage. One implication of this fact is that web applications have a lower barrier for use than desktop applications since they do not require users to download something before they can start. This is not true for web applications that rely on browser plugins, unless the latter can be assumed to be installed already, which today means Flash or maybe the Java plugin. This does not mean that web applications provide the only viable option, but in many situations they constitute the first exposure that the learners may later depart from with more dedicated desktop- or mobile applications.

The mobile platforms have a somewhat different situation, since much effort has gone into making it easy to find and install mobile applications via various application stores, such as Apple's Appstore and Google Play. But unless the application is to be provided exclusively on a mobile platform, it is still important to consider web applications as well. Especially since many web applications can be made cross-platform from the start, ranging from traditional desktop environments to mobile platforms.
Another reason for focusing on web applications is that the web has become the natural place to seek information, communicate, and collaborate. As learning applications often include handling information, communicating and collaborating, by analogy, it is natural for users to expect a web-based solution for learning applications as well.

Figure 3 presents an outline of a possible categorization of applications. If we ignore the catch-all native application type, all the other types mentioned above are intended to be networked applications. Furthermore, the application types are intended to follow the client-server architectural constraint according to the classification established by (Fielding, 2000).

Native Applications are usually installed directly in an operating system and make use of graphical user interface (GUI) frameworks such as MFC, Cocoa, Swing, GTK+, etc. in order to provide a unified look-and-feel. Each native application stores its data locally and can usually not be run on other operating systems without extra development efforts.

RPC Native Applications are native applications that rely on the Web Architecture for interacting with backends, but not on a web runtime environment such as a web browser for generating the user interface. That this category is named RPC Native applications simply means that the application uses RPC-style web services to interact with underlying services. Hence, just like SOA mentioned above, RPC Native Applications utilize, but do not integrate with, the Web

---

29 MFC is the Microsoft Foundation Class Library in C++.
30 Cocoa is a framework for developing native application on OS X.
31 Swing is part of the Java Foundation Classes and contains an API for developing graphical Java applications.
32 GTK+ is a multi platform toolkit for creating graphical user interfaces in C and C++, originally developed for Linux.
Applications Types

Architecture. Today, thanks to the higher demands on access from multiple devices and the cost of maintaining several architectures in parallel, many native applications are turning into RPC Native Applications, at least if they require network access.

**Web Applications** rely on a web runtime environment such as a browser to render the user interface based on technologies such as HTML, CSS, and JavaScript. All users that have Internet access can potentially use these applications; since the data is usually stored on the server. Web applications can be further subdivided into static webpages, progressively enhanced, RPC Ajax, and RESTful Ajax web applications.

**Static Web Pages Web Applications** was the first type of web applications that emerged when web developers gave sets of web-pages a unified look-and-feel and introduced user interaction via HTML forms. The distinguishing criteria with respect to the progressively enhanced, Ajax, and RESTful web applications are the lack of any advanced use of JavaScript to achieve dynamic behavior outside of page reloads. The static web-page model remains a valid candidate for web application development, especially when the demands for reliability and support for old or simple clients is a requirement.

**Progressively Enhanced Web Applications** are web applications built using the progressive enhancement technique that was introduced in 2003, see (Champeon & Finck, 2003). The basic premise is to provide a user interface based on simple web pages that should work in all browsers, even simple browsers such as those provided by non-smartphone mobile environments and screen readers. However, in more capable browser environments, the web pages are enhanced via scripting techniques in order to provide a richer experience. A vital part of the progressive enhancement technique is to design web pages cleanly and to use the markup of HTML to emphasize the meaning of the content. For example, utilizing the "H1" element for headers rather than a semantically free "DIV". This will help simple browser environments such as screen readers to present the content correctly as well as help developers to address the right piece of content when enhancing the experience via separate styles and functionalities. The approach is sometimes referred to as *semantic html* and even though the author has found it hard to locate the origin of the term, many of its principles can be traced back to the W3C's Web Content Accessibility Guidelines (Chisholm, Vanderheiden, & Jacobs, 1999). Hence, progressive enhancement, if done right, provides improved accessibility.

**RPC Ajax Web Applications** are a form of rich web applications that provide functionality without exposing the user to page reloads. The common principle of RPC Ajax web applications is that they communicate with the server in the background via RPC calls and change the web page via scripting technologies when new data is received. The result is a web application that has a look and feel that is much closer to desktop applications and often makes use of richer user interaction principles such as drag and drop, dialogs, notifications etc. The term Ajax was introduced in 2005, see (Garrett, 2005). Ajax was originally AJAX which stood for Asynchronous JavaScript And XML but today also includes other scripting technologies as well as other formats for data retrieval. The reliance on RPC for interaction with underlying services makes this category of applications similar to the RPC native applications. Hence, as described above, they utilize the web architecture but do not integrate well with it.

**RESTful Ajax Web Applications** are similar to RPC Ajax web applications with the difference that interaction with the server is done via RESTful web services rather than via RPC. This category both utilizes, and integrates well with, the Web architecture and can be designed to work with Linked Data.
RESTful Native Applications are native applications that use RESTful web services. They have a strong relationship with RESTful Ajax web applications. In fact, web applications that need to be ported to other devices such as mobiles or tablets could be built as native applications relying on the same RESTful services that the web application relies on.

Clearly, from the descriptions above, all the application types except two types of native applications utilize the Web Architecture. If we also exclude the RPC Native Applications and RPC Ajax Web Applications, we are left with those application types that integrate with the Web Architecture, that is, those that use the linked character of the web.

3.8 Summary

In this chapter we have seen how a range of technologies are related to each other. The starting point has been the Web Architecture and how the Semantic Web data relates to the Web Architecture. We have also seen how SOA utilizes, but does not integrate well with, the Web Architecture, which makes it a poor choice when working with Semantic Web data. In figure 4 an attempt has been made to visualize the relations as arrows. Since SOA only utilizes the Web Architecture, the arrow is labeled with an ‘U’ as well as emphasized by a dashed line. The other architectures and technologies both utilize, and integrate with, the Web Architecture, and hence the relations are shown as arrows labeled with ‘U&I’.

The figure also shows the application types introduced in section 3.7 and how they relate to the various architectures and technologies. First, ROA supports application types that are based on RESTful principles. Second, SOA supports the two RPC-based application types. Third, the static web pages and progressively enhanced web applications rely on the Web Architecture directly since they, by definition, work only with data already included in the page. That is, they
are not allowed to make use of any web services, neither RPC style or RESTful, in order to request additional data. All these relations to the application types are shown as arrows labeled with 'S', indicating which architecture that supports each application type.

There are also a few relations around Linked Data that in figure 4 for simplicity is shown as support relations. First, a support arrow indicated that Linked Data relies on Semantic Web for its expression. Second, two support arrows points from Linked Data to the RESTful applications indicating that Linked Data fulfills enough of the REST architectural constraints (globally identifiable resources with retrievable representations in standardized formats that connect to each other in a hyperlinked manner) to be easily consumed by RESTful applications.

In principle, Linked Data can be consumed by other application types as well. One possibility is to have a server-side proxy that provides indirect RPC-style access to Linked Data. However, this is not the natural approach, since it either requires a specific proxy for each situation or a generic solution that would resemble a reimplemention of the web as a web service. Hence, there is no arrow between Linked Data and the RPC-based application types in the figure. Another possibility would be to have an arrow between Linked Data and progressively enhanced web applications, since Linked Data can be embedded within web pages using RDFa. However, as discussed in section 3.6, this is in general not a fruitful approach due to conflicting needs between how to design a web application and how to design a reusable web service. Hence, no arrow there either.

Finally, let us address the lack of arrows between Semantic Web and other application types than the RESTful application types (implicit from Semantic Web supporting Linked Data). The argument in this chapter is that if the integration with the Web Architecture is weak, then the integration with Semantic Web will also be weak. This is why there are no arrows from Semantic Web to the RPC-style applications. Furthermore, the argument above, stating that embedding Linked Data in web pages is not always the best approach when developing progressively enhanced web applications, is valid for Semantic Web as well. Generally speaking, Semantic Web data can of course be included in web pages in a human readable format, which would indicate some form of consumed by relation. However, this is a too generic feature, since it applies to all other data expressions as well and does not mandate a consumed by arrow to either static web pages or progressively enhanced web applications from Semantic Web.
4. Two Semantic Web based Learning Applications

In this chapter we will introduce two Semantic Web based Learning Applications. Each application will first be introduced and after that the added value of relying on Semantic Web technology will be discussed. The final section will introduce three obstacles that have been identified during the development of the applications.

4.1 EntryScape - a Personal and Collaborative Portfolio Suite

EntryScape is a web application that has been developed in close supervision and in part by the author since 2001 and gone under different names such as efolio, SCAM Portfolio, Confolio and finally EntryScape. The web application has been discussed shortly in paper 2, 3, and 6. EntryScape is an Ajax web application that depends on RESTful services provided by the accompanying back-end solution EntryStore, see section 6. However, since the following sub-chapters will not go deep in technicalities on how the system was implemented we will use EntryScape to refer to them both.

4.1.1 The purpose of EntryScape

Originally EntryScape was supposed to be used in strictly educational scenarios, as a hybrid between an archive for teachers and a personal portfolio for learners. Due to the iterative development, today there are a number of other scenarios that EntryScape supports. However, the educational scenarios have received continued support and have been vital in driving the design and development forward. In the educational scenarios under consideration, both teachers and learners were given their own virtual space, an e-portfolio, where they were able to upload files and collect resources in the form of links. The resources were then organized into folders,
which could be shared selectively in order to enable collaboration, or shared publicly for anyone to see. The teachers and learners could link to each others' material in order to reuse it - or contextualize it by providing additional information.

There are many perspectives on the use e-portfolios. For instance, in (Lorenzo & Ittelson, 2005), a distinction is made between student, teaching and institutional e-portfolios. Another perspective considers whether the primary purpose of the e-portfolio is to support the collection of the learner’s work, support deep learning via reflection and other techniques, or be more oriented towards assessment of the learner, see (Barrett & Wilkerson, 2006). EntryScape has not been explicitly developed to target any of these perspectives, although the use cases have largely been centered on supporting the collection of work material - for both the learner and the teacher - as well as on supporting collaboration.

Much emphasis has also been placed on standards and the portability aspect. Learners should be able to "take their portfolios and leave", that is, move their portfolio from one provider to another. For instance, this can be useful when a student ends her studies at a university and starts to work in a company. Her transferred e-portfolio can be a useful asset, both for showing established skills as well as providing a good tool to support her continued learning.

Today there are many e-portfolio systems as well as more general-purpose collaboration platforms. Two of the more popular ones are Google Drive and Dropbox. Let us list a few features of EntryScape that are not present in either of these. EntryScape has support for:

1. handling metadata beyond simple title and description.
2. linking to web material.
3. handling resources that are not documents.
4. handling relations between resources.
5. defining groups that can form communities and be used for access control.

A more comprehensive comparison with currently available systems is out of scope for this thesis, although a rough categorization of systems that are related to the underlying EntryStore is carried out in section 6.2.

4.1.2 How EntryScape works

In a typical installation of EntryScape, users are given portfolios where they can organize information. The approach taken is that information is captured in entries which pair resources with corresponding resource information, that is, metadata. Each entry also contains other useful administrative information such as access control, date of creation etc. A large part of EntryScape's flexibility is the wide range or resources allowed. For example, resources can take the form of uploaded files, linked material on the web, physical objects in the world like the Eiffel Tower, or less tangible objects like the concept of a circle, or a fictional character like Superman.

33 Google Drive, formerly known as Google Docs, is a platform for managing files, most prominently known for the support for real-time authoring of text documents, presentations and spreadsheets. See http://docs.google.com/.
34 Dropbox is a platform for storing and sharing files. The most well-known strength of Dropbox is that it keeps the files synchronized across various platforms and devices. See http://dropbox.com/.
This flexibility allows even the portfolios themselves, folders, users, and groups to be expressed as entries in EntryScape although they do receive special treatment by the system to enforce special rules, for example that the folders must form a strict hierarchy without loops.

When a user enters the EntryScape application he can choose to visit a range of portfolios as well as user and group profile pages. When navigating to a specific portfolio the topmost folder will be shown with its contained entries displayed in a list, see figure 5. Depending on the nature of the resource described by the entries, there will be links to web pages, download options for uploaded files, or sub-folders that can be navigated into etc. Selecting an entry yields a more detailed view. The detailed view (on the right) shows the resource information at the bottom and an embedded view of the resource at the top, in figure 5 a video from YouTube is embedded and in figure 6 a presentation from SlideShare is embedded. If no embedded view is possible an representative icon will be shown to convey the resource's character. Furthermore, some resources may not have a digital representation and can neither be embedded or accessed separately, in such situations the detailed view will be the main attraction.

To provide an overview, there is a location bar that shows the path to the current folder and within which portfolio it belongs (such a location bar is often referred to as breadcrumbs). In figure 6, the location bar shows that we are looking at the "Top" folder in the portfolio named "YouTube - Viticulture".

In figure 6 we see that it is also possible to bring forward a tree of the folder structure when needed. This is done by clicking the leftmost icon in the location bar. Figure 6 also shows how the folder view can be changed to display entries as icons instead of rows in a list.
Each entry may be kept private or shared with others. Hence when a user logs in, he might see additional entries that have been shared with him or to some group where he belongs. In the profile page all portfolios and folders to which a user has special access are listed for better overview, see figure 7.

Figure 7: Profile view of the author of the thesis with information on participation in groups, which portfolios and folders he has access to and recently added or modified material.
To create new or modify entries in a portfolio there are context menus available over each entry in the folder listing (as indicated by the drop-down symbol after each entry in figure 5). Which kind of resources - as well as which metadata that can be provided - is configurable in EntryScape. Figure 8 shows a dialog where metadata is provided for a resource according to the IEEE/LOM standard.

4.1.3 Added value of Semantic Web

In section 2.5 it was claimed that RDF can help people to communicate. EntryScape largely realizes this claim by:

1. Allowing anything to be talked about by referencing a wide variety of resource types, including uploaded files, existing resources on the web physical resources etc.

2. Flexibility in which terms that are used by relying on an editing framework, see section 5 on RForms and other metadata editing frameworks, that replaces development effort with a set of configuration steps. This allows administrators of EntryScape installations to reuse existing terms in new configurations that was not originally foreseen as well as create new terms when needed.

3. EntryScape can be configured to allow users to create new terms in the form of SIOC concepts and reference from other entries. This way of creating new terms is somewhat limited but avoids the configurations steps that are better suited for administrators with knowledge of metadata standards.

In section 2.3 it was discussed how RDF can support objective metadata. In EntryScape there is already support for a range of metadata standards such as IEEE/LOM and Dublin Core via the RForms configuration mechanism. Furthermore, when a file is both uploaded and described in EntryScape, the metadata will have the same origin as the resource and will therefore be
perceived as authoritative. However, for links, or large installations with many users it will be a question of trust. EntryScape does provide provenance information (author and contributors, date of creation etc.) as a necessary piece of the puzzle.

Another important feature of EntryScape is that entries can contain both user provided metadata and reference external metadata. This is especially useful when contextualizing resources by complementing the authoritative metadata. For example, a user can re-purpose a generic resource in a learning context by adding metadata that provides the learning aspect that was not present in the authoritative metadata.

In general, referencing authoritative metadata improves the quality of the information in the system by making information available that would otherwise not be present. It also minimizes the risk of mistakes when manually providing the information. References to external resources and their resource information is especially useful when integrating with existing search services such as library systems that have rich metadata already.

Finally, section 2.4 discussed how RDF supports subjective metadata. The whole idea of giving users their own portfolios where they can keep their own semantic web data is fundamental to providing subjective metadata. But also the support in EntryScape to express various forms of comment, ranging from simple rating to a well structured review. The Comments are by themselves represented as entries in EntryScape and kept in the commenter’s own portfolio. This both strengthens the richness of the user’s own portfolio as well as allows the user to return to old discussions easily.

4.2 Conzilla - a Concept Browser

Conzilla (Naeve, 1999) is a Java application that has been developed since 1999. The first implementation of Conzilla relied on XML documents for representing context-maps and the concepts and concept-relations. The second version of Conzilla replaced XML with RDF in anticipation of the benefits of Semantic Web for learning applications as discussed in section 2.6. It also reduced the amount of specific solutions in favor of reusing established standards and vocabulary, especially with respect to how to reference and describe resources. Conzilla version 2 is discussed in paper 4, while earlier versions were discussed shortly in paper 1 and 2. Conzilla has been gradually improved over the years and today has a lot of features, some of which go beyond the scope of this thesis. The following subsection will briefly explain how concept browsing works and how Conzilla supports it. The last subsection describes how relying on Semantic Web technologies has made Conzilla a better tool for learning.

4.2.1 The purpose of Conzilla

Conzilla is an implementation of a so called concept browser (Naeve, 2001b), which aims to create overview of an information landscape without losing depth. Overview and depth are achieved by creating maps that display concepts and their relations at different level of detail. The maps can be horizontally connected by using the same concept in different maps or vertically connected by zooming in on the details of a concept by providing a detailed map.
A concept browser can be used to clarify your own thoughts on a thorny topic, which corresponds to systemic learning\textsuperscript{35}. Or, it can be used as a collaborative tool to clarify conceptual similarities and differences among the participants. It can also be used as a presentation tool where the order of the presented content can be chosen at the time of presentation.

An important consequence of the reuse of concepts is that the maps can be connected and gradually expanded by collaboration with others. This supports a continuously evolving human discourse that can result in an ever growing network of knowledge (by connecting new information with old ideas in a way that does not force consensus). See section 2.5 and 2.6 for a longer discussion of how this relates to learning applications and Semantic Web.

\subsection*{4.2.2 How Conceptual Browsing and Conzilla Works}

Conceptual browsing allows you to investigate contexts-maps, concepts (including concept-relations), and content. A context-map graphically presents a selection of concepts and concept-relations that ties the concepts together. Every context-map, concept and concept-relation should come with at least some descriptive information (metadata) for example name, description, target group, purpose etc. To support multilingual context-maps, parts of the metadata expressions containing string values can be translated into one or several additional languages. In addition to metadata, concepts and concept-relations may be enriched by adding relations to content, often in the form of web resources. Each content resource should be described with metadata as well, although typically with other metadata fields than those used for concepts, concept-relations and context-maps. In figure 9 a context-map shows a UML-style\textsuperscript{36} diagram of the relations between natural-, whole-, rational-, real-, and complex numbers. The figure shows both metadata (a floating semi-transparent box) and content labels (sidebar on the right with green background) for the ‘whole number’ concepts.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{context-map.png}
\caption{Context-map showing different types of numbers. For the whole numbers concept, metadata is shown as a popup and content labels are shown on the right.}
\end{figure}

\textsuperscript{35} According to David Hestenes, systemic learning means that concepts derive their meaning from their place in a coherent conceptual system. For further details see [naeve et. al. 2005].

\textsuperscript{36} UML, Unified Modeling Language, is a visual modeling language in the field of object-oriented software engineering.
A context-map may be built in a single piece, but it may also be an aggregation of parts:

- **Layers** – if a context-map becomes too complex it might be beneficial to break it down into different layers with the basics in one layer and extra information in additional layers. There is a layer panel where visibility of the layers can be managed.

- **Contributions** – it might seem strange at first, but anyone is allowed to contribute to a context-map by including additional concepts or concept-relations, or for that matter providing additional metadata. Just as for layers, there is a contributions panel where the origin of the contributions can be seen and their visibility managed.

Figure 10 shows a context-map with 6 layers where the user has selected the second layer from the top in the layer panel which has resulted in corresponding concepts and concept-relations being marked in the context-map (in a red/pink color).

Figure 11 shows a context-map which has both layers and contributions. Just as for layers the visibility of contributions can be controlled by the user. A user that finds an interesting map may chose to make a contribution and if he so prefers, publish it for others to find, see (H. Ebner, Palmér, & Naeve, 2007) for a deeper discussion on how collaboration around context-maps is achieved.
Also note that the context-map in figure 11 uses another modeling style, in this case it is the Dialog Mapping style (Conklin, 2005) that has been mimicked. Finally, there are two mechanisms that allow users to navigate between context-maps:

- **Concepts appear in different Context-maps** – A concept (and concept-relation) is not owned by a context-map, instead it is an independent entity which might be included in many maps. Hence, from a single concept it is possible to detect a list of context-maps wherein it occurs, this is sometimes referred to as the contextual neighborhood of the concept (or concept-relation).
- **Context-maps can contain hyperlinks to other context-maps** – In each context-map a concept or concept-relation occurrence may have a hyper-link to another context-map.

In figure 12 it is shown how it is possible to navigate to another context-map via the contextual neighborhood of a concept. Note that the selected option "Different types of numbers" is the context-map shown in figure 9 and 10. Figure 12 also shows the URI of the current context-map, a tool-bar with icons for zooming, navigating back and forward in the conceptual browsing history, changing language, bringing forward layers and contributions etc. There is also the option of switching to edit mode, via the pen icon, that allows the user to modify his own context-map or, if the context-map was initiated by someone else, make a contribution to it.

---

**Figure 11**: A context-map which, in addition to layers, also has contributions from others than the original author. The contribution is selected and shown in a red/pink color. The map is shown with another color-theme that uses a white background which works better with the Dialog Mapping modeling style which contains both icons and thin dotted borders around the concepts.
The figures in this section (figures 9-12) are taken from the internal modeling work of the research group of the author and they are selected because they demonstrate clearly how Conzilla works. For examples of how Conzilla has been used within an European learning network, see (Maillet, 2008).

4.2.3 Added value of Semantic Web

Before we go into the added values of using Semantic Web for Conzilla, we shortly describe how RDF is used to represent concepts, concept-relations, content and context-maps. First, a concept is a resource in an RDF graph that has been brought forward by the context-map. Second, a concept-relation corresponds to a triple in the RDF graph that connects two concepts appearing in the context-map. Third, content components are resources connected to concepts via properties recognized as content relations. Fourth, context-maps consist of sets of layout resources that provide concepts or concept-relations with a graphical appearance. Note that a context-map may collect concepts, concept-relations and content from more than one RDF graph. Furthermore, the context-map itself may be distributed into several RDF graphs. A more complete description of how RDF graphs are used for representing context-maps can be found in paper 4.

Let us now briefly revisit the benefits listed in section 2 starting with how RDF helps people to communicate. The following three characteristics of RDF was identified to be important for an evolving human discourse, see section 2.5. This is how Conzilla and context-maps supports them:

- Talk about anything - a context-map allows anything to be exposed as concepts, concept-relations if it is a relation, or content and be described with metadata as well exposed relations between concepts.

More specifically the Network of PhD students in the TEL research area created within the PROLEARN Network of Excellence was operational between 2004 and 2008.
• Unlimited set of terms - new concepts, concept-relations and content may be formed at any time and be given meaning by relations and descriptive metadata.

• Reuse terms in new contexts - concept, concept-relations or content may be used in different context-maps with different purposes.

Second, section 2.3 states that RDF can be used to express objective metadata:

• Established metadata expressions - the properties used to describe concepts, concept-relations and content may be chosen from established metadata standards such as Dublin Core or IEEE/LOM to increase interoperability. The SHAME/RForms metadata authoring framework is used in order to produce correct expressions. See section 5 for more details.

• Exposing the origin of metadata - when a context-map is assembled from different sources there is a mechanism to separate each contribution visually and investigate its provenance. The provenance includes origin, date of creation, and additional meta-information that the author included during the publishing phase to aid in understanding the purpose and trustworthiness for the corresponding information.

Third, in section 2.4 it is stated that RDF supports the expression of subjective metadata.

• Re-purposing existing knowledge - since context-maps allows multiple RDF graphs to be combined it can be used to provide a subjective view on existing knowledge.

• Flexibility in collaboration - context-maps can be created by a single author, assembled from a range of independent contributions, or built collaboratively.

In addition, by relying on a standard for knowledge expression, the information expressed by Conzilla can be re-used by other applications without special knowledge of what a context-map is.

4.3 Identifying major Obstacles

The applications EntryScape and Conzilla did not materialize overnight, in fact they are the result of many months, or even years, of development, testing and also more theoretical concerns. Naturally, some parts of the development went smoothly, while other parts turned out to be real obstacles that needed separate attention before the work on the application could continue. Three obstacles stand out more than others and are introduced below.

4.3.1 Obstacle 1

Both EntryScape and Conzilla have been hindered by the need to both present and edit a wide range of Semantic Web data. Both applications target non-experts (people that have little or no knowledge of RDF, the format for expressing RDF, or for that matter issues around metadata interoperability) and need to provide user friendly solutions for editing metadata. Both applications initially had metadata editors that were hard-coded to specific vocabularies, but as new requirements was formulated more flexibility was needed. Let us summarize the obstacle in a single sentence:

Lack of non-expert and user friendly solutions for presenting and editing Semantic Web data that is not hard-coded to use a specific vocabulary.
Both applications depend on SHAME/RForms as mentioned in paper 3, 4, and 6 as well as discussed in some detail in the implementation chapter in paper 5. See section 5.5 for a longer treatment of how the solution has emerged over time from a range of different needs.

4.3.2 Obstacle 2

Development of EntryScape has been hindered by the need of a stable platform for working with resources and their metadata. From the start there was a need to share material with others, search effectively, link as well as upload resources, keep track of when, what and who did something etc.

Development of Conzilla has been hindered by lack of good solutions for managing related content resources and support for collaboration. A solution for collaborating around Context-maps was developed. Although the idea of how to achieve collaboration is sound, the actual implementation is too specific and will be hard to maintain. Management of content resources is yet not realized in Conzilla.

The obstacles from EntryScape and Conzilla share a common ground which can be roughly summarized in the following sentence.

Lack of solutions that can handle both private and collaborative management of resources together with related Semantic Web data.

The application chapter in Paper 3 and the use case chapter in paper 6 both describe how SCAM/EntryStore has been motivated by the needs of digital portfolios/Confolio/EntryScape. See section 6.3 for a longer treatment of how SCAM/EntryStore have been developed over time in response to changing needs and new technologies maturing. Future development of Conzilla is expected to integrate with a solution such as EntryStore, which is also mentioned in the conclusions of paper 4.

4.3.3 Obstacle 3

Both EntryScape and Conzilla have been developed iteratively where dependencies to various technologies have changed in response to new insights. In Conzilla the switch from application specific XML to RDF and from a fixed metadata editor to a configurable metadata editor are representative of this process. In EntryScape the switches from triplestore to quadstore and from static web page applications relying on server-side templates to RESTful Ajax Web Applications compatible with Linked Data are representative.

All these changes of technologies correspond to new insights of improved approaches for integrating with Semantic Web Technology, as well as for making better learning applications. To change from one technology to another is often a big step that requires a sacrifice of previous effort. Not having the insights needed, or not gaining them quickly enough is clearly an obstacle for development. Although new technologies will be developed resulting in new and perhaps better ways of developing learning applications, it would be beneficial to have a set of recommendations that guide software architects and developers when building learning applications based on Semantic Web technology. The obstacle in a sentence:

Lack of recommendations for how to build learning applications based on Semantic Web technology
5. Presenting and Editing RDF

The focus of this chapter is how to overcome the first obstacle:

*Lack of non-expert and user friendly solutions for presenting and editing Semantic Web data that is not hard-coded to use a specific vocabulary.*

To overcome this obstacle we first provide an overview of four different categories of tools that can be used for presenting and editing RDF data. We will see that the first two categories, syntax- and ontology-based tools do not provide viable solutions to overcome the obstacle. This is due to the fact that they are neither targeting non-experts nor are they as user friendly as would be needed. The last two categories, graph- and form-based tools, require deeper treatments and are discussed in separate sections. The following two sections focus on configurable RDF forms. The first section gives a brief discussion of related initiatives. The second section presents six iterations of development on the SHAME/RForms framework for configurable RDF forms during the period 2002 to 2011. For each iteration, new features, validation and lessons learned are discussed. The final section of the chapter provides a summary.

5.1 Tool Categories for Presenting and Editing RDF

We start by looking into existing tools for presenting and editing RDF data. To provide some structure, the tools have been grouped into four categories, of which the two last ones will be further analyzed in the sections below. The categories are based on which approach the solutions take for modifying the RDF data. They are:

**Syntax-based**

Perhaps the most common way to edit RDF manually is to use a text editor such as Emacs. If the focus is on the RDF/XML format, XML editors can be used, although they do not provide much guidance as the XML Schema for RDF/XML is quite vague. Neither do syntax-based tools meet the user friendly or the non-expert requirement in the first obstacle.

**Ontology-based**

When the need arises to create an ontology, many authors turn to specific tools that hide the specifics of how to express an ontology in RDF. One of the most commonly used tools is open source ontology editor Protégé\(^{39}\). There are also commercial alternatives such as Altova SemanticWorks\(^{40}\). Both tools allow authoring of ontologies as well as instances of these ontologies. However, this does not cover all kinds of RDF data. One such non-covered example is given by RDF data that is expressed with the help of Dublin Core terms. As indicated by its name, this type of RDF data is a set of terms rather than an ontology. If Dublin Core was to be transformed into an ontology, just to allow ontology editors to work better with RDF data that uses it, it would impose restrictions on Dublin Core terms that would exclude many of the scenarios that it is aimed to support. In general, ontologies are useful for well-specified domains, but they are not always the best choice for broader, widely reusable vocabularies.

Furthermore, ontology tools are primarily targeted towards people that need to author ontologies rather than learners or teachers that just need to provide/annotate some data. Hence, ontology tools do not meet the user friendly and non-expert requirement of the obstacle either.

**Graph-based**

Tools that visualize RDF data as graphs are plentiful, often relying on graph visualization libraries such as Graphviz\(^{41}\). However, tools that also allow editing of the RDF data visually are less frequent. IsaViz\(^{42}\) is perhaps the most widely known graph tool for editing RDF data. Conzilla, as described in section 4.2.3, can also be used for editing RDF data, although this is not its first priority of use.

Whether graph-based tools for presenting and editing RDF data are good enough with regard to the non-expert user friendly requirement of the obstacle can be discussed. A longer treatment follows in section 5.2 below. Let us just observe that there are often parts of RDF data that it is better to show in lists, tables or trees views than in a graph view.

**Form-based**

Finally, tools that expose RDF data in forms are perhaps the most common. There are initiatives ranging from vocabulary-specific HTML forms to fully configurable RDF Form solutions. Foaf-a-matic\(^{43}\) is a typical example of vocabulary-specific HTML form, since it allows users to create a personal profile expressed in the Friend Of A Friend vocabulary (FOAF). SAHA metadata editor, see (Kurki & Hyvönen, 2010), on the other hand uses the vocabulary definitions together with specific directives in a separate XML file to configure the form. The latest version of SAHA, SAHA 3, is at its base a progressively enhanced web application as the RDF data is fetched from the triplestore and transformed into HTML on the server side. The SHAME/RForms framework has large similarities to SAHA 3 using vocabulary definitions in combination with a dedicated configuration. But they differ both on expressibility of the configuration and on the architectural level where the latest iteration, RForms, has evolved into a Javascript library that can be embedded in different kinds of Web applications. The iterations of SHAME/RForms will be discussed in more detail in section 5.5.

The Fresnel display vocabulary, see (Pietriga, Bizer, Karger, & Lee, 2006), is an interesting initiative for presenting RDF that unfortunately cannot easily be generalized to the editing case\(^{44}\). Basically, Fresnel allows a set of graph patterns, lenses, to match parts of RDF graphs according

---

39 http://protege.stanford.edu/
40 http://www.altova.com/semanticworks.html
41 http://www.graphviz.org/
42 http://www.w3.org/2001/11/IsaViz/
43 http://www.ldodds.com/foaf/foaf-a-matic
to a predefined algorithm. After a set of lenses has captured all RDF data to present, each lens provide instructions for how to present the matched data, including layout and stylesheet information.

The form-based approach seems to have the possibility to support non-experts in a user friendly manner, that is, at least the requirements for the obstacle seem to be fulfilled. In section 5.3 below the form-based alternatives are discussed in more detail.

5.2 Presenting and Editing RDF in Graph Based Interfaces

One of the fundamental differences between RDF and XML is that RDF is a language that can express graph data (with directed and labeled arcs), while XML provides a tree-based data structure. Hence, it is tempting to suggest a graph-based interface for presenting and editing RDF. The IsaViz tool provides graph-based editing, including clever mechanisms for zooming, and laying out large graphs. However, there is also functionality for suppressing parts of the RDF graph and display it in a form-like manner instead via the GSS (Graph Style Sheets) introduced in (Pietriga, 2003). This is quite useful if there are properties that have string values rather than referencing other resources.

Conzilla is also a tool that allows presentation and editing of RDF graphs, although indirectly. There were attempts to allow Conzilla to expose entire RDF graphs, making both blank nodes and strings visible in the Context-map. However, those attempts were abandoned for two reasons. First, there are technical limitations regarding how to uniquely reference blank nodes and literals (IsaViz has the same problems but overcomes them by copying the RDF graph into a separate more capable format than RDF/XML), see paper 4 for a deeper treatment. Second, it did not make sense from a usability perspective to clutter Context-maps with parts that were better presented in a form-like manner. The solution for Conzilla was to show only resources identified by URIs (concepts) and statements where both subject and object were resources identified by URIs (concept relations). The rest of the RDF graph is shown as information around concepts in a form that can be brought forward when needed, the only exception being labels that are shown inside of the concept boxes.

Presenting and editing parts of RDF graphs in graph-based interfaces can be done in many ways, and the approach taken in Conzilla is discussed in depth in paper 4. It is quite clear though, that graph-based interfaces can never provide the only solution to the outlined obstacle, even if they successfully fulfill the non-expert and user friendly requirements. The reason is simply that graph-based interfaces do not integrate well in the overall design of many other applications. For instance, the majority of web applications provide input of data via forms and a graph-based interface would stand out too much and perhaps draw attention away from the main activity. It is also a question of accessibility and efficiency in producing the RDF data. For instance, it is not obvious how to input data in a graph-based interface using solely the keyboard.

Furthermore, from a more intuitive standpoint, graph-based interfaces often have a different feel to them than editing data in a form. They indicate simplicity and invite people to experiment in a way they would not do if the data was in a form. Hence, if the application developer does not want to indicate that the RDF data is less formal and correct than other information it is perhaps

44 An attempt was made as part of the LUISA project to develop an editing extension to Fresnel, the approach is outlined in the first version of Deliverable 3.2: Annotation Profile Specification. The approach was later abandoned and the next version of the deliverable focus on a separate specification corresponding to iteration 4, that is, SHAME2.

45 Tree-based data structures are similar to graph-based structures with the added constraints that there cannot be any loops and there must be at most one in-bound arc to each node.
better to stick with form-based solutions. One exception to this is when the RDF data to edit has a truly relational character, and is also a crucial part of the application. In this case graph-based interfaces make more sense. Consequently, we now focus our attention on how to present and edit RDF data in forms.

5.3 Presenting and Editing RDF in Forms

There are many established solutions that expose information in forms, including pure HTML forms and XForms. The majority of these solutions rely on an underlying information model - a specification of fields, vocabularies to use, validation of data etc. - to be available when applied in a specific setting. The information model serves as input to developers when constructing the form. If the information model changes after the form is finished, a developer has to be involved again to make sure that the latest version of the information model is accurately reflected in the form.

The information model for XML is the XML info-set, which is a tree of elements with attributes. Hence, a form for editing any kind of XML would be very generic. XML editors often allow users to narrow the XML instances they want to edit by specifying an XML Schema. The XForms W3C recommendation goes one step further by providing a standardized way for developers to restrict to a smaller set of XML instances and foreseen interactions by using a combination of XML Schema, the XForms model structure, and specific form controls.

Likewise, the unrestricted information model for RDF leads to very generic forms that expose the statements of an RDF graph flatly or possibly as a set of interconnected trees. Another possibility is to develop a presentation/editor form that is targeted towards a specific expression in RDF, for instance the Dublin Core set of terms. These two approaches are referred to as fixed respectively generic categories of RDF presentation/editor forms in paper 5 (where they are called annotation tools).

However, neither approach is satisfying. First, the generic form approach does not provide enough guidance for regular users, although perhaps it is useful for experts. Second, the fixed approach does not fit well with the aims outlined in section 2 of this thesis, since the expressiveness and flexibility gained by using RDF would be hindered if a developer must be involved every time the need arises to express something new.

What is needed for RDF is configurable forms, similar to XForms as a configurable mechanism to construct forms for XML. There are substantial differences though. Although XForms can have a focus on xml instances that often coincide with entire documents, a configuration for RDF must match sub-structures from a graph. Furthermore, the practice to mix vocabularies in new ways, for example when expressing Linked Data, requires a mechanism to express restrictions valid in a certain context that the existing vocabulary definition languages OWL and RDF Schema are not suitable for. In addition, there are needs such as providing order among properties, suitable labels, cardinality restrictions, expressing form controls etc. Since the requirements are many and sometimes quite detailed, we group them into categories. The following four categories are taken from paper 5 although with a slightly different wording:

---

46 XForms is a W3C recommendation to incorporate forms for editing XML in other markup languages such as XHTML, ODF or SVG, see (Boyer, 2009).
### Presenting and Editing RDF in Forms

<table>
<thead>
<tr>
<th>Category</th>
<th>Description of category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>includes support for editing arbitrary RDF, including support for datatypes, language literals, blank nodes, RDF containers and collections. However, every RDF graph does not necessarily fit in a single form, especially when there are loops in the RDF graph. The configuration mechanism of the form should specify which parts of the RDF graph to edit/present and which to leave untouched.</td>
</tr>
<tr>
<td>Structure</td>
<td>includes cardinality constraints and order of selected RDF graph structures. A direct correspondence between the graph structure and its presentation in the form should not be enforced. For example, it should be possible to hide a complicated graph structure with intermediate resources from the end-user, and it should be possible to introduce pedagogical/cosmetic categories when the graph-structure is too flat.</td>
</tr>
<tr>
<td>Interaction</td>
<td>includes hints on how to choose values from vocabularies/ontologies, e.g., check-boxes, radio-buttons, drop-down menus, or search-dialogs. It also includes mechanisms for string validation according to datatypes, control of auto-complete mechanisms etc.</td>
</tr>
<tr>
<td>Presentation</td>
<td>includes multilingual labels and descriptions to aid the user in deciding how to edit. Font, color, indentations, borders, and everything else that has to do with appearance is also included here.</td>
</tr>
</tbody>
</table>

#### 5.4 Related Initiatives for Configurable RDF Forms

First, let us note that XForms is not an option for configurable RDF forms, simply because RDF is not XML. Concretely, there is no canonical expression of RDF in RDF/XML. Hence, to use XForms, even for a very restricted vocabulary, would require a preparation step where perfectly valid RDF/XML would have to be rewritten into a specific expression just so that a specific XForms-based editor would understand it. However, XForms is still an interesting and relevant technology to gain inspiration from when looking at configurable RDF forms.

Now let us briefly consider RDF Schema, OWL and DSP (introduced below) as possible configuration mechanisms.

**RDF Schema**

Let us consider the completeness category introduced above. RDF Schema provides a way to define vocabularies in terms of classes, properties and their domains and ranges. In the best scenario, an RDF Schema would model a specific domain quite well and allow forms to be generated to edit class instances. But in order to edit RDF data that contains generic properties such as those defined by Dublin Core terms, the RDF Schema does not help. See also the discussion about ontology editors in section 5.1 above. Furthermore, there are also problems with RDF collections and containers, since there is no way in RDF Schema to indicate what the restrictions are of the member resources (for instance whether or not they are all instances of a single class).
For the other categories, structure, interaction and presentation, RDF Schema provides nearly no guidance except for labels and descriptions. It is important to notice that RDF Schema is not worthless from the configurable RDF Forms perspective. It does contain relevant information, but, it needs to be complemented.

**OWL**

OWL, Web Ontology Language, is a more powerful language than RDF Schema although its main objective is the same: to define classes and properties and how they relate. From the perspective of configuring forms the only added value is that OWL provides cardinality restrictions on properties.

**DSP**

DSP, Description Set Profiles, is an initiative from Dublin Core to formalize application profiles, see (Nilsson et al., 2007). DSP provides a way to describe how various properties should be used in a specific context. It does so by listing which properties that should be used together, and it provides restrictions more specific than those given when the properties were originally defined.

Regarding the completeness category, DSP is defined in terms of the Dublin Core Abstract Model, but there is a mapping to RDF which is reasonably complete. Hence, DSP manages to describe most RDF data with a few exceptions such as RDF containers and collections.

In the structure category DSP provides cardinality restrictions just as OWL does, it also provides information on which structures that should correspond to separate forms, and which parts that should be kept within a single form. This goes beyond what OWL provides. But in the interaction and presentation categories, DSP is weaker than RDF Schema and OWL, since it does not provide any labels or descriptions.

From this we note that none of the above initiatives on their own, or even in combination, covers the requirements of the configuration mechanism we seek. However, as they do contain relevant information that should not be duplicated, a configuration mechanism should only complement them with additional information, and not replace them. For instance, the SAHA metadata editor is a good example of this where the configuration mechanism is a combination of an RDF Schema with a custom XML file containing additional information used to generate the user interface.

We now briefly look at SAHA before diving into the details of the development of SHAME/RForms. To start with, SAHA edits instances of classes, hence to some extent it has the same limitations as an ontology editor with regard to editing instances. However, this seems to be a limitation of how its configuration mechanism is triggered, rather than a fundamental limitation of the system. One of SAHA strengths is in the interaction category where it can indicates when to use a search interface, when to load alternatives from an ontology, when to allow local instances to be created, etc. On the other hand, SAHA is not as capable when it comes to the structure category. It does not support cardinality restrictions and what is shown in the user interface is always in direct correspondence with the expression in RDF. In the presentation category, SAHA draws information from RDF Schema to generate appropriate labels and descriptions. Additional styling is achieved by changing the server side templates, that is, outside of the configuration mechanism.

[47](http://dublincore.org/documents/dc-dsp/)
5.5 Six Iterations Towards Configurable RDF Forms

The following subsections outline six iterations of development of configurable RDF forms ranging from 2002 to 2011. Each iteration corresponds to a more or less mature phase in the development process which has been validated in one or several real settings that are outlined in a validation section. From each iteration experience from development, deployment, testing, adaption to specific needs etc. are summarized in a "lessons learned" section.

The author has been involved in the design and development of all iterations except the first. Interested readers are welcome to investigate iteration 2-5 at the sourceforge page for SHAME: http://sourceforge.net/projects/shame/ and iteration 6 at the google code page for RForms: http://code.google.com/p/rforms/.

5.5.1 SCAM Portfolio Metadata Editor version 1

The first steps towards a configurable metadata editor was developed as part of the SCAM Portfolio. The aim was to provide a range of appropriate metadata editors for different types of resources. The type of metadata supported was only direct property value pairs that had a corresponding RDF expression. The configuration for an editor was provided by a Java class that specified an ordered list of properties together with the information whether the property pointed to a literal or a resource, if multiple values were allowed, and if the input field should allow one or several lines of text. The rendering was performed using JSP (Java Server Pages is a templating mechanism for generating web pages). The editors provided in the default installation contained a few combinations of Dublin Core properties.

Validation

The first version of the SCAM Portfolio and its metadata editor was deployed and used in teaching at two departments at Uppsala University. The teacher education department used the system as a way to help the future students gain better knowledge of how to use technology in teaching. The department of archiving and library science used the system both as an archive system and as e-portfolios for the students.

Both departments gave feedback on usability and technical shortcomings of the system, especially with respect to how to maintain the system in a longer perspective, for example with respect to backup and administrative user interfaces.

Lessons learned

Since the configuration step required writing code, this version was, in practice, a fixed RDF editor, although the first steps had in fact been taken towards making it configurable. However, since changing the metadata editor required involvement of developers, the flexibility of using RDF was not yet materialized.

Furthermore, the added value of reusing established vocabularies to achieve interoperability was also quite weak, since there was no interaction with other systems. The only benefit was regarding compliance with standards, but since the import/export was not realized, the benefit was only theoretical.
The simple metadata representation with direct properties from the resource quickly became a limitation of the system. For instance it did not allow metadata according to the IEEE/LOM standard to be expressed. Hence, support for editing more complicated structures quickly became a requirement.

5.5.2 The SHAME library

To address the shortcomings of the SCAM Portfolio metadata editor, a separate reusable Java library was developed see (Eriksson, 2003). The library was later named SHAME, which stood for Standardized Hyper Adaptable Metadata Editor. The new library was improved in several areas:

- **Introducing a new configuration mechanism** - The mechanism introduced for configuring RDF editors relied on the combination of a RDF *query* and a *form template* that were connected via variables. The configuration mechanism is explained in (Eriksson, 2003).

- **Introducing a format for the configuration mechanism** - The query was represented using a simplified version of the RDF representation of the Edutella Query Language, see (Nejdl et al., 2002). Note that SPARQL did not exist at that time. The format resembled the reification mechanism in RDF, except that some of the subjects, predicates and object resources were considered to be variables rather than fixed values. A new RDF expression, based on RDF Collections, was introduced in order to capture the internal structure of the form template.

- **Query by example** - Since the SHAME configuration mechanism consisted of a query, it was a natural step to allow an editor to act as a query form. First, a user selected a suitable SHAME form, second, he filled in some of its fields, third, the edited, and now more specialized, query was executed (for instance sent out on the Edutella network), and finally, the results were nicely formatted using the original SHAME form.

- **Multiple views** - In accordance with the MVC\(^\text{48}\) pattern, the model was separated from the view. The model, termed *form model*, was introduced as a layer on top of the result of executing the query. This made it easier to develop several functionally equivalent views, utilizing different UI-rendering techniques.

- **Java Swing-based view** - By utilizing a range of ready made components from the Java Swing library, both an editor and presenter was developed. In figure 13, the Swing-based view of SHAME is used to edit a piece of content using the IEEE/LOM configuration.

- **Editing SHAME forms with SHAME forms** - A special SHAME form was developed that could edit arbitrary (non-recursive) form-templates. Since a form-template is a tree with unknown depth, this required a recursive construction in both the query and form-template. In principle it was possible to edit the query in a SHAME form, but it was rather awkward. Hence, the RDF-editing capabilities of Conzilla were used instead.

---

\(^{48}\) Model-View-Controller is a design pattern.
**Validation**

The SHAME editor was integrated into Conzilla where it replaced the ImseVimse fixed metadata editor\(^49\) that was hard-coded to work with IEEE/LOM. A few different metadata configurations were created for concepts, concept-relations, content, and context-maps.

A search interface to the Edutella network, the SHAME-consumer, was also developed. It provided users with a list of pre-created queries, each appearing as a form which could be partly filled out in the query-by-example style, see (Eriksson, 2003). The results received were displayed with the same form.

During the work with the Swedish Educational Broadcasting company (UR) four media pedagogues marked up around 10 000 media items using a desktop application based on SHAME. Specific effort was spent on interaction with the SCAM 2 repository, see section 6, and allowing media items to be connected via drag-and-drop into the SHAME form.

Finally, the SHAME configuration mechanism was used to formalize the first RDF expression of IEEE/LOM, see (Nilsson et al., 2003).

**Lessons Learned:**

The introduced configuration mechanism successfully allowed editing of tree-like RDF structures. Other RDF structures certainly exist, but they are not easily handled in form-like interfaces. Hence, the restrictions of SHAME regarding what it can edit were, already from the

\(^{49}\) \url{http://sourceforge.net/projects/imsevimse}
start, quite reasonable, and they have not been challenged later. A major improvement was the separation between form-model and the view that rendered the form, since it both turned out to be quite natural and also allowed great flexibility in construction of views.

The management of many SHAME forms was somewhat awkward, since it involved keeping pairs of RDF files together.

To edit SHAME forms with SHAME forms, although possible, turned out to be a bit too complex and fragile, especially with regard to the graph part. This pushed the development away from the RDF representation of QEL and in a longer perspective away from having the query separate from the form-template. However, this was not realized until the last iteration, RForms.

5.5.3 SCAM Portfolio Metadata Editor version 2

The second version of the portfolio metadata editor incorporated the SHAME library and extended it to make it more useful in a web setting. The following improvements were made:

- **Formlets for managing forms** - To simplify form management, formlets were introduced in order to give an identifier, name, and description to a query and form-template pair. The formlet could also contain references to vocabularies needed by the form-template.

- **Aggregating formlets** - The process of joining RDF editors together required a careful merging process where both the queries and the form-templates were merged in parallel. With the introduction of formlets, the management and formalization of this merging process was simplified.

- **Mapping formlets to types** - A single editor does not fit all situations. Hence, a way to trigger different formlets for different type of resources was introduced. The types were required to be organized into a subclass hierarchy. Consequently, for a given resource of a specific type, a SHAME editor could be constructed by aggregating formlets specified for the given type and all of its super-classes.

- **HTML Form-based view** - A JSP-based editor and presenter was developed that transformed the form model into a HTML Form. Structural changes, such as duplicating or removing structures, required page loads that performed operations on the form model and then re-rendered the page.

**Validation:**

Despite the fact that the system had undergone a major rewrite the portfolio installations at two departments at Uppsala University were upgraded with a few minimal conversion steps. The new more capable metadata editor could also edit the old RDF data without problems. This was strong indication that the foreseen added value of RDF, with good choice of vocabularies as a stable and interoperable language, was indeed realized.

The new system was also installed at KTH, where around 400 media students were given e-portfolios over the course of several years. Several experiments were undertaken during this time, for example with new types and metadata expressions for describing concepts. This was made possible by the configuration mechanism of SHAME which allowed experimentation without changing the base system.
Originally the online media library for UR (the Swedish Educational Broadcasting company) had a hard-coded website that collected out specific metadata fields. This media library was also made available in the portfolio view and hence via the new web-based metadata editor. Experiments were made where teachers and students reused material and provided their own metadata on top of the authoritative metadata authored by UR.

Lessons Learned:
It became simple to configure a wide range editors by writing many small formlets capturing individual triples or small structures starting with a single outgoing property, for instance dc:creator pointing to a blank node with outgoing triples.

The configuration mechanism was now powerful, but no longer simple. Hence, there was a need for proper documentation by SHAME-form authors.

The HTML Form-view via server-side languages such as JSP required a lot less effort to maintain than the Java Swing-based view. It also integrated better with other web frameworks due to the power of CSS. However, the page reload during the editing process proved to be an irritation.

5.5.4 SHAME 2
SHAME 2 was a partial rewrite of the original SHAME library that, in addition to a clean-up of the code, tried to unify terminology and provide documentation:

- **RDF graph always correct** - A dependency tree is introduced as part of the matching/editing engine so that it reacts to user interaction and immediately inserts or removes parts of the graph if the validity of the latter is changed.

- **SPARQL support** - The QEL language option was gradually phased out in favour of SPARQL queries. A distinction from regular SPARQL semantics was that the OPTIONAL modifier was assumed on all paths from the root. It could therefore be left out, which simplified the writing of the queries.

- **Annotation Profile specification** - The SHAME-form configuration was renamed "Annotation Profile". The name was intentionally close to the established term Application Profiles. In addition, the query part of the Annotation Profile was now referred to as the graph pattern.

- **Formulator: An Annotation Profile editor** - A Java application was developed that could edit the Form template and graph pattern of a formlet. It could also be used to assemble formlets into compound formlets.

Validation:
This version of SHAME was the basis on which paper 5 was written as well as the more formal description of Annotation Profiles as deliverable 3.2 in the LUISA project, see (Palmér, Enoksson, & Naeve, 2007). Fredrik Enoksson has also written a licentiate thesis on the subject (Enoksson, 2011).

The improvements in this iteration were mostly preparatory and did not lead to the development of new end-user applications or deployment in new scenarios. However, the formulator was very appreciated among those who developed Annotation Profiles, since it made the process of constructing Annotation Profiles both less time consuming and less error prone.
5. Presenting and Editing RDF

Lessons Learned:
A major improvement was the use of a dependency tree for keeping the graph minimal and correct at all times. This turned out to be useful for debugging purposes, for better user interface design, since auto-save could be better supported, as well as for simplified integration with other software components.

The support for recursive queries was abandoned, since it gave rise to very complicated checks in the dependency tree.

The term Annotation Profile did cause some confusion since the term "Annotation" has slightly misleading connotations. It indicates appending or commenting on something which is more specific than the general activity of providing statements on resources which is what RDF offers. If so needed RDF can support annotation or commenting, but then only by using specific vocabularies with appropriate semantics.

Explaining how Annotation Profiles works was not a simple task. The possibility of having different formats for both the graph-pattern and form-template is a nice feature, but it also complicated the design by introducing additional terminology.

5.5.5 Ajax-SHAME

Up to this iteration, SHAME was a library that had to be bundled with the application developed. Now the effort was made to also expose SHAME as a service.

- **Annotation Profile service** - This service made Annotation Profiles easily accessible over HTTP in a specific JSON format. Upon requesting an Annotation Profile, the service would load all dependencies and build a single Annotation Profile. This also included pre-calculating and including all choices needed from the vocabularies inline in the format.

- **Form-model service** - This service exposed form models over HTTP in a specific JSON format. To request a form-model you had to specify which Annotation Profile to use, which resource to edit, and where to find the corresponding RDF.

- **Separating form-model service from RDF storage** - An attempt was made to allow the form-model service to communicate with remote RDF storage solutions. Solutions like SPARUL\(^50\) and plain RESTful updates of named graphs were tested.

- **Javascript-based view** - A Javascript library was developed. From a form-model and an Annotation Profile it rendered the user interface via DOM manipulations. The editing process modified the form-model directly in the client and communicated with the form-model service via Ajax requests.

Validation:

Within the LUISA project\(^51\) two testbeds, one at a department at Université Henri Poincaré and the other within a division of the EADS aviation company, made use of Ajax SHAME to update their Learning Object Metadata repositories. The repositories were maintained independently and access was only provided via an extension of the SPARQL protocol. This separation was crucial in order to allow parallel development within the project. The approach of using Annotation

---

\(^{50}\) SPARUL or SPARQL/Update, an update language for RDF, was a member submission to W3C in July 2008, [http://www.w3.org/Submission/SPARQL-Update/](http://www.w3.org/Submission/SPARQL-Update/) it has been superseded by further development of SPARQL.

Profiles to configure the editor was also crucial since it allowed changes to the ontology quite late in the project. In the end, only a few hundred learning objects were authored. This was considered a success since the project aimed to demonstrate new technology rather than broad roll-out and uptake.

Ajax SHAME was also ported to Confolio/EntryScape and used in both the project Organic Edunet and HNet, where, in total, several thousands of resources were edited. After some initial and easy-to-solve problems with browser compatibilities, Ajax SHAME was successfully used to edit quite extensive amounts of metadata on resources, metadata that also made use of large vocabularies.

**Lessons Learned:**

The Annotation Profile service needs to support managing and creating Annotation Profiles. This will avoid having to publish the Annotation Profiles on specific URLs or update the entire service just to update an Annotation Profile.

The separation of the form-model service from the RDF-storage turned out to be more complicated than anticipated, especially since protocols like SPARUL was not mature enough. At the time the approach with remote update of entire named graphs seemed to be the best alternative, since the alternatives sometimes tend to pollute the RDF storage, see discussion in (Enoksson, Palmér, & Naeve, 2007).

By sending form-models back and forth, the business logic for SHAME is in part duplicated on the client side, which should be avoided.

### 5.5.6 RForms

The experience from the earlier iterations clearly demonstrated the need to simplify. Hence, RForms, short for RDF Forms, was produced as a more or less a complete rewrite in JavaScript. The configuration mechanism and the terminology was also simplified:

- **New configuration mechanism** - The configuration mechanism, termed RForm-templates[^52], was introduced as a structure in JSON. The mechanism resembles the form-template structure but with inline information about properties and constraints that was previously contained in the graph pattern.

- **JavaScript RDF API** - A Javascript API that provides utility functions for working with RDF/JSON. Including a simple statement search and a cross-graph statement assertion.

- **JavaScript matching/construction engine** - The engine uses the RForm-template to match parts of an RDF graph and as a result produces a tree of bindings. These bindings can be considered to be a kind of instantiation of the RForm-template, which to a large extent resembles the form-model from SHAME. But it also incorporates the characteristics of the dependency tree, since it is responsible for keeping the RDF graph minimal.

- **Match-all mode** - It is possible to load a range of RForm-templates and for a given RDF graph ask the system to generate a RForms-template that covers as much of the graph as possible. This is useful for instance when the RDF data originates from another system and vocabularies are mixed in an unknown manner. The match-all mode resembles how Fresnel works, although it also allows editing. Note that it is possible to combine the

match all mode with a required RForms-template making sure that certain parts of the form is always available. This is especially useful when editing a resource from scratch as the form would otherwise be empty.

**Validation:**

Ajax-SHAME has been replaced with RForms in EntryScape and the installations for the projects Organic.Edunet, Hnet, Voa3r, and TelMap have been updated without any significant problems.

A converter has been developed to semi-automatically generate RForm-templates from various RDF Schemas, including Dublin Core, FOAF and Schema.org. Another converter combines DSP expressions with RDF Schema information to produce enriched RForms-templates. These conversions were accomplished much quicker than previous conversion attempts in SHAME, since the RForms-templates expressions are easier to work with than previous expressions.

**Lessons Learned:**

By moving the matching/construction engine to the client side, the form-model format and the form-model service could be removed altogether. Hence, RForms can now run without server-side support, which is demonstrated on the google code page. This makes the integration with other web applications much more smooth, since the only requirement is to include the JavaScript, provide a RForms-template and use the API to launch forms for presenting or editing RDF data.

Furthermore, by merging the graph-pattern and the form-template the two-step process of matching a RDF graph to a form-model was reduced to one step. The resulting code-base is smaller and easier to maintain. This simplicity out-weights the benefits of having a clean graph-matching step based on SPARQL.

RForms needs to better define the API it exposes to the web application it is embedded in. It also needs to better support relations between resources exposed by the web application for instance by triggering search dialogs or support cut and paste between the surrounding web application and text fields in RForms. It is also a challenge to present such related resources that are only referenced with a URI in the graph being presented. Suitable labels for related resources are often maintained in other RDF graphs that will need to be loaded separately to avoid showing URIs to related resources directly in the form.

There is still a need for a service where RForms-templates can be constructed and assembled into various constellations (the Annotation Profile Service mentioned in the previous iteration was never complete and it does not work well with RForms-templates). The service must support collaboration in the form of copying or assembling RForms-templates from other RForms-templates. The need has only to a small extent been alleviated by converters from other formats.

There is also a need for a service that can generate choices (based on constraints in the RForms-template) from established vocabularies, since this is probably a bit to computationally expensive to do in Javascript on the client side, especially for large vocabularies.
5.6 Summary

In this chapter we have gone through various ways to edit and present RDF data. The requirements of non-expert and user friendly solutions made us discard the syntax and ontology-based solutions. Graph-based solutions remains a possibility, but perhaps not the preferred choice when developing learning applications, especially not in a web environment. Hence, the focus on form-based solutions, more specifically on configurable form-based solutions. When going through related initiatives, there were neither any existing configuration mechanisms, nor actual solutions that were found to be good enough. Consequently, the focus shifted to six iterations of the SHAME/RForms framework, which was developed with the requirements of configurable forms in mind.

The three first iterations are mainly about expanding the capability and flexibility of the editors, proving that a configurable editor did provide enough value in comparison to static and generic editors. The fourth iteration, SHAME 2, was about maturing the framework, while the last two iterations focused on simplifying and moving to a more lightweight web environment.

With this drive towards simplicity, certain functionality has been sacrificed. For example, recursive forms, query by example, the Java Swing view and the separation between graph-patterns and form-templates.

The major features of the framework that has been accomplished, refined, and which remain in the last iteration described above can be summarized as:

1. A configuration mechanism that is easy to understand and work with.
   Introduced in iteration 2, improved in iteration 4 and 6.

2. Configurations that can be assembled from smaller dedicated configurations, sometimes referred to as formlets.
   Introduced in iteration 3, improved in iteration 4 and 6.

3. Support for match-all mode, where many configurations, formlets, are combined to match as much as possible of a given RDF graph.
   Introduced in iteration 6.

4. Model and view separation, which makes it possible to have multiple presentation- and editing views.
   Introduced in iteration 2, improved in iteration 6.

5. RDF expressions are always minimal and correct according to the configuration.
   Introduced in iteration 4.

6. A stand-alone Javascript library that is easy to integrate in web applications, taking an RDF graph and a configuration as input.
   Introduced in iteration 5, improved in iteration 6.

There are also features that have been requested but not yet realized. The first and most important is an envisioned service for managing RForms-templates. Second, a service for generating choices from large vocabularies. Third, an established protocol to remotely update RDF repositories. Although, from the perspective of RForms, this is the responsibility of the surrounding web application, it would still be beneficial to have a stable protocol, since it would make API design easier\(^53\). Fourth, better support for managing relations to other resources so that

\(^{53}\) The SPARQL 1.1 Graph Store HTTP Protocol mentioned in section 3.6 may be the right protocol for the task.
RForms can work better with the surrounding web application. Here RForms can take inspiration from the configuration options used by the SAHA metadata editor. However, since RForms does not have its own dedicated RDF storage service as SAHA has, these configuration options will probably look a bit different.
6. Read/Write Resource and RDF Framework

The focus of this section is on how to overcome the second obstacle identified in section 4.3:

*Lack of solutions that can handle both private and collaborative management of resources together with related Semantic Web data.*

This chapter will start by breaking down the second obstacle into requirements that can be addressed individually. Second, it will look at existing solutions and why they are not good enough. Third, the four iterations of SCAM are presented, especially how the consecutive improvements closely zoom-in on overcoming the obstacle. Fourth, the chapter concludes with a summary of the findings.

6.1 Breaking Down the Second Obstacle

We start by clarifying the second obstacle by breaking it down into a set of requirements. Whether a single system, or a set of components in a larger platform, will meet these requirements does not really matter. However, for simplicity we will assume that the requirements apply to a platform. The requirements are:

1. **Manage any RDF** - Since the vision that has helped to identify the obstacle states that applications should allow people to express themselves on a growing range of topics, it is not enough to support a fixed and limited set of properties. Hence, the requirement is to allow any RDF to be expressed.

2. **Manage Chunks of RDF** - preferably in a manner that is suitable for describing individual resources. The chunks must be easy to access without hindering queries that span multiple chunks. Keeping track of provenance of each chunk is also important. This requirement is further elaborated in papers 3 and 6.
3. **Manage resources** - some of the resources that are addressed via URIs will have digital representations that need to reside somewhere. Clearly, some of the digital representations already exist on the web and cannot be managed by a single system. But those resources that are provided by the users themselves, like uploaded pictures or text documents, need to be stored somewhere. Providing support for managing these resources in the same platform that holds the corresponding RDF-graph has several benefits, including enforcement of access-control and ease-of-use by not forcing people to switch systems and keep them synchronized.

4. **Manage web linking** - URIs are needed for allowing linking to both resources and chunks of Semantic Web data. This requires coining new unique URIs when appropriate and tying the resource and the appropriate chunk of Semantic Web data together via links. This will allow regular people, not only knowledge representation specialists, to express themselves in a scalable manner.

5. **Manage private and shared information** - at its core the platform must support individuals and groups in managing information. For instance, this means that each person and group should have a place to store information. Furthermore, fine-grained access control mechanisms should provide a cornerstone for forming collaboration around both resources and associated Semantic Web data.

These requirements are not exhaustive, but they follow reasonably well from the vision, and the perceived obstacle, and they also correspond to the needs observed during the four iterations of development.

### 6.2 Existing Solutions

Today there are many systems/platforms that are relevant in the sense that they meet several of the requirements. For example, to make an attempt to find all that match for example two or more of the requirements would be a daunting task. Instead, we will take the approach of focusing on categories of systems/platforms, and by looking at representative examples we will see which of the requirements they fulfill, see table 5 for an overview.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Triplestore</th>
<th>DAM</th>
<th>ePortfolio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Manage any RDF</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Manage chunks of RDF</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Manage resources</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>4. Manage web linking</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5. Manage private and shared information</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
**Triplestores**

Triplestores are perhaps the most obvious category, since they all provide a way to manage RDF graphs via programming language specific APIs. In addition, most of the modern triplestores such as Apache Jena\(^{54}\), Sesame, and Virtuoso, also provide access via protocols such as SPARQL over HTTP. Clearly, by design, triplestores fulfill the first requirement of managing any RDF. The second requirement of managing chunks of RDF is supported by those triplestores that provide support for named graphs and an accompanying protocol for managing the named graphs. For example the SPARQL Update or the alternative RESTful SPARQL Graph Store HTTP Protocol could be used. The other three requirements are not supported at all.

**Digital Asset Management**

Digital Asset Management, or short DAM, is a category of systems that ingest, store, annotate, catalog, publish and distribute digital assets. A digital asset is often considered to be images and other multimedia but could also include documents and other resources that have digital representations. Hence, DAM systems fulfill the third requirement of managing resources. Since they also provide support for publishing the assets, they do provide URIs and can be said to have reasonable support for the fourth requirement of web linking.

With respect to metadata, DAM systems are often quite rigid with support for only a few bits of information that should be handled manually, such as title and description, together with more administrative information such as size, modification date and version control. In addition, a DAM system provides specific support for the metadata it can extract from the assets, such as the EXIF tags found in images. A few systems are more flexible, such as Fedora Commons that allows new sets of metadata to be supported and by default has support for Dublin Core. It also uses RDF internally to encode the relations between the assets, and the same is done by the Nuxeo DAM. Still, neither the first nor the second requirement on managing RDF can be said to be supported in the DAM systems of today, since there is no generic support. There is hope for the future here though, since the Apache Stanbol project\(^{55}\), described as a set of components for semantic content management, attempts to provide semantic services that can be used to enhance content management systems.

The main use case of DAM systems is to support organizations in managing their assets, either for use internally in the organization or externally to communicate with customers or the public. Hence, the fifth requirement of managing private and shared information is seldom fulfilled.

**ePortfolios**

The final category we will take a look at is ePortfolios. The focus of ePortfolios systems is to maintain a private archive of material that can be used for work and learning, for reflection on progress, and for showcasing achievements. Many systems offer the possibility to configure different views for showcasing in different settings. The distinction with respect to a DAM system is the focus on the individual and the communication among people. This includes functionality to collaborate, comment, assess etc. Hence, ePortfolios fulfill the requirement of managing private and shared information. In addition, they also fulfill the requirements of management of web links and resources just like DAM systems but in a less elaborate manner.

---

54 Jena was initially developed by HP (Hewlet Packard) but was converted into an Apache project in 2012.
6. READ/WRITE RESOURCE AND RDF FRAMEWORK

The ePortfolios of today, like Mahara\(^{56}\) and desire2Learn ePortfolio\(^{57}\) do not support RDF. However, for the quite recent Leap2A specification for ePortfolio interoperability\(^ {58}\) - which is supported by Mahara - a Semantic Web version of the specification called Leap2R is in progress.

6.3 Four iterations

In the following sections four iterations of development are discussed. The requirements introduced above are addressed more or less from the start. An exception is the weak support for managing RDF in the first iteration. The focus is on how the requirements have been implemented, especially on problems and weaknesses.

The direction of development has been guided by attempting to overcome the focused obstacle in two ways. First, by providing a specific solution, an ePortfolio that helps people to create and manage resources and their Semantic Web data, both privately and in collaborative settings. Second, by trying to establish a platform upon which a range of such solutions can be built.

The author has been involved in the design of all iterations and heavily in the development of the last iteration. The interested readers are welcome to investigate iteration 1 and 2 at the sourceforge page for SCAM: \[http://sourceforge.net/projects/scam/\]. Iteration 3 has unfortunately no publicly accessible code repository. Iteration 4 is accessible at google code under the EntryStore project: \[http://code.google.com/p/entrystore/\].

6.3.1 SCAM 1 - efolio

The first incarnation of SCAM, sometimes referred to as the efolio, was built with the aim to be something between a personal portfolio and an organizational archive for material. The typical scenario was to support teachers in providing uploaded or linked material to their students. The students had their personal portfolio as well, and they could collect the material they needed from the teacher's portfolio or from elsewhere.

The system was built using java servlets and JSP (Java Servlet Pages) technology so that it could be deployed in any servlet container. Underneath, it relied on a database for the metadata and other structured information and a filesystem to store uploaded files. Even though RDF was used internally for storing metadata for each resource, the platform only exposed the 15 Dublin Core properties. A resource could be either a link, an uploaded file, or a folder.

In addition to the HTML view of the efolio, the folder structure was exposed via WebDAV\(^ {59}\) for supporting mounting the efolio as a networked folder. This would allow better integration with the desktop by supporting things like drag-and-drop, and access to the efolio material from within desktop applications such as word processors.

Validation

The first version was deployed and used in teaching at two departments at Uppsala University. The teacher education department used the system as a way to help the future students gain better knowledge of how to use technology in teaching. The department of archiving and library science used the system both as an archive system and as personal portfolios for the students.

\(^{56}\) [http://mahara.org/]
\(^{57}\) [http://www.desire2learn.com/]
\(^{58}\) [http://www.leapspecs.org/2012/2A/specification.html]
\(^{59}\) WebDAV is an extension of HTTP to support collaboration in editing and managing documents on HTTP-servers.
Both departments gave feedback on usability and technical shortcomings of the system, especially with respect to how to maintain the system in a longer perspective, with respect to for example backup and administrative user interfaces.

Observations

Two important - and early recognized - limitations was the lack of good search and backup/restore capabilities. Neither limitation was addressed in the first version, due to the choice of using RDF for the representation, which brought along a new set of challenges.

WebDav PROPFIND method was the first interoperability problem encountered, since it required a mapping from simple Dublin Core to the live properties of WebDAV. A real nuisance with the WebDav protocol was the need to choose a single name for a resource, when, for instance, there could be several titles in different languages. The solution was to introduce a special property that stored a name for a resource that was set initially with the title, but which had to be modified independently later. This also solved the problem of not allowing illegal characters in URIs.

The architecture was not very flexible, for instance the user interface was generated via JSP that communicated directly with the database by means of a small help library. Furthermore, the web.xml mapped directly from URIs to the JSP driven pages that were supposed to answer each request. This turned out to be a bit inflexible when the application needed to be changed or new separate user interfaces were required. As an additional consequence, there was no separate access to the data, for instance via RMI60 or Web Services.

Finally, the editing framework used, the SCAM Portfolio metadata editor v1, had severe limitations and only included support for simple RDF structures.

6.3.2 SCAM 2

SCAM 2 was more or less a rewrite where much effort was spent on scalability, and modularity. Paper 3 introduces SCAM 2 and discusses it in detail. The following list outlines the main advancements:

- **Switch to a mature triplestore** with a mature triplestore (the Jena package was chosen, which at the time was supported by HP) the amount of triples that can be managed efficiently improved. There is also a wider range of ready-made functionality to rely on, for instance RDF graph search via several different query languages.

- **SCAM Records and Contexts** introducing an algorithm to collect triples together by starting from a resource, and following the natural direction of triples until a non-blank node is found. SCAM Contexts corresponded to a set of SCAM records that need to be managed together, in fact coinciding with the portfolio concept. Access control was provided on both records and contexts.

- **Enterprise Java** provides an environment for enterprise software, an environment where large-scale, multi-tiered, reliable, and secure network applications can be built. SCAM 2 exposed an API as session-beans with bean-managed persistence that connected to the underlying triplestore.

- **Modular architecture** was mainly achieved by introducing a configurable workflow engine, the **SCAM controller**, that allowed a range of reusable commands to be composed into command chains. A chain ended by handing over to JSPs that generated views that

---

60 Remote Method Invocation is the object oriented approach for doing RPC in a java environment.
relied on information assembled earlier in the chain. Hence, when creating new applications, it is often enough to configure new command chains from existing commands and provide new JSP pages to generate the views of the application. Furthermore, to simplify development of the JSP pages, a range of specific taglibs were provided to act as a set of UI building-blocks.

- **Simple and advanced search** in the form of free-text search and more advanced graph-queries. Both search options could be executed either against single portfolios or against the entire repository, and they were realized using a modified implementation of the RDQL language\footnote{RDQL - a query language for RDF, see \url{http://www.w3.org/Submission/RDQL/}, it was later superseded by SPARQL.} that respected access control rules.

- **All data in the graph.** In addition to storing the metadata for a resource, also the folder structure, access control and provenance information were stored in the RDF graph. With this change, only uploaded files, user- and group definitions as well as login credentials were kept separate from the graph. The folder structures were expressed using RDF collections in a manner which was supposed to be compatible with the IMS Content Packaging Model\footnote{A zip file with a manifesto in XML introduced by the IMS consortium, see \url{http://www.imsglobal.org/content/packaging/}.}.

- **Improved metadata editing** was achieved with the SCAM Portfolio Metadata Editor version 2, as introduced in section 5.5.3 above. Several editors were provided per default, for instance based on Dublin Core and on IMS metadata to complement the IMS Content Packaging expression of folders.

- **Backup/restore functionality** for each portfolio was accomplished by storing all metadata, that is, metadata for every file, link, folder etc. in a single large RDF/XML file together with a directory containing all uploaded files. This folder structure could be used for reverting back to an earlier state or moving the portfolio to another installation.

### Validation

SCAM 2 was used as part of the Folio thinking project at KTH where around 400 media students were given personal portfolios over the course of several years. Many of the students used the system as a convenient way to transport files from their home computer to the school. Hence, one of the questions that was raised repeatedly concerned the continuity of the installation, that is, for how long time could the students rely on the system being available.

SCAM 2 was also used as a basis for the online media library for the Swedish Educational Broadcasting company. Four specialists worked full time to curate information about educational programs on radio and television using a specialized desktop tool that connected to the repository. The result was shown as an advanced search interface, where the public could search via a range of specific semantic search options. In a more experimental approach, specific themed collections were constructed by a range of experts who re-purposed material by linking to material in the public part of another portfolio. Plans to offer this functionality to the public, for instance targeting teachers, were unfortunately never realized.

### Observations

The installations showed that an RDF backend could be used in real settings with a quite high load. However, a number of scalability issues remained, mainly due to the way the triplestore was used:
• How to do free text searches efficiently. Initially, SCAM used RDQL which pulled up large parts of the database into memory just to perform substring matches on literals. Even though this improved in later versions of Jena, with pushing more of the queries to the database, the performance needed to be checked carefully. Still, as can be seen in later chapters, to support good text search, a separate solution like Apache Lucene\(^3\) that maintains a dedicated index is a necessity.

• Querying several portfolios efficiently at the same time. Keeping all portfolio data in a single graph was not acceptable since it would lead to conflicting metadata expressions, or that only one person could express something about a given link. Keeping the portfolios separate was the only solution, although it required substantial changes and a deviation from how Jena used relational databases for storing RDF. The effect was that it was hard to upgrade and therefore diminished the benefits of relying on a stock triplestore.

• How to bring up all metadata for a single resource efficiently. Since SCAM 2 relied on Jena, which had no fourth column to group the triples together, bringing up all related triples required several requests to the database until all relevant triples had been found. This problem is discussed in detail in paper 3.

In addition to these scalability aspects, the modularity improvements also had their problems.

• Moving to the enterprise java platform introduced a lot of complexity at a time when the standard was still young. Some parts of it was beneficial, such as the security model, concurrency control and support for remote procedure call. But important parts, such as the use of entity beans and transactions in distributed applications, were hard to realize due to the graph-like character of the underlaying data model, as exposed by the triplestore.

• Deployment of SCAM in an Java EE application server, JBoss 3.2.1\(^4\), turned out to be both complicated and a bit fragile with many manual steps, especially when upgrading. Whether the reason was lack of maturity in the application server, lack of documentation or inexperience by developers and system administrators in handling application servers does not really matter. What does matter is that people, both within the development project and external parties, were uncomfortable with the deployment process.

• The introduction of a controller layer, the SCAM controller, greatly simplified development of new applications. However, maintaining a controller framework proved to be a substantial task in itself. It is therefore better to rely on existing libraries such as Apache Struts\(^5\), or Open Symphonys WebWorks\(^6\), since it would likely lower the amount of code to develop and make the library more focused.

Overall, the focus on a more modular design in SCAM 2 allowed for many different applications to be realized in theory, although in practice the system was hard to install and maintain.

6.3.3 SCAM 3

The third iteration of SCAM encompassed a shift toward pragmatism, that is, SCAM 3 should be easy to install, configure to specific needs, and maintain.

---

\(^3\) [http://lucene.apache.org/](http://lucene.apache.org/)

\(^4\) [http://www.jboss.org/](http://www.jboss.org/)


• **No more enterprise Java** - the architecture of SCAM changed to depend only on Java servlet containers rather than full-blown application servers. Since very little new code had to be written to support this change, it indicated that for SCAM the added value of relying on Java EE application server functionality was small and easily replaced with a few key libraries.

• **Established controller framework** - the SCAM controller was replaced with the OpenSymphony Webwork controller framework. This both reduced the amount of code to maintain in SCAM and provided richer and more reliable functionality.

• **Change of templating language** - the Java Server Pages templating language was replaced with Velocity, which forces a stricter separation between view and business logic, since it does not allow generic inline code. This stricter separation made it easier to reuse business logic when building new applications.

• **Separation of repository and application** - yielding better flexibility in deploying applications upon a single repository. In situations where no new business logic was required, new applications could be realized without programming. It was enough to provide new velocity pages, localization files, and specify which metadata to use.

• **Simplified metadata configuration** - to allow quick setup of new applications, a simplified metadata editing framework was created that only supported a subset of what SHAME offered.

• **Support for metadata federations** - to participate in metadata federations support for harvesting protocols such as OAI-PMH\(^67\) and a variant of SQI\(^68\) called Fire, see (Paulsson, 2009), was introduced.

• **Feed support** - all folders were exposed via RSS, and, if the content was appropriate, also as Podcasts.

**Validation**

The SCAM 3 based portfolio application was installed at Umeå University and used in a few courses. At the time of writing, around 500 students have portfolios in the installation.

The repository part of SCAM 3 has been used as a base for several other applications. For example, a search service for science and technology, a search service for math, and a generic embeddable federated search service for school resources in Swedish. The separation of repository and application turned out to be useful for quickly trying out new application ideas, which could later be carefully tweaked to meet specific needs.

**Observations**

The pragmatism worked in the sense that with the new design, applications could be quickly built and more easily maintained. To a large extent, SCAM 3 showed that it was possible to overcome large parts of the obstacle. However, at the time when SCAM 3 had matured, the world had moved on and it was clear that new challenges had arisen:

• Ajax could be used to build more appealing web applications and mashups. It just required that the data needed to be exposed somehow.

\(^{67}\) [http://www.openarchives.org/pmh/](http://www.openarchives.org/pmh/)

\(^{68}\) Simple Query Interface
• Exposing Semantic Web data via SPARQL endpoints and as linked data was now the recommended practice.

• References to external metadata and enhancement of harvested metadata required a more advanced model.

6.3.4 SCAM 4 - also known as EntryStore

SCAM 4 was a complete redesign and has since been renamed to EntryStore as a reaction to the focus on entries. The new design also included the use of Named Graphs and the exposure of the Semantic Web data as read/write Linked Data. The ePortfolio application was completely redesigned as a RESTful Ajax Web application and now termed EntryScape to match with EntryStore. Below are listed some of the central characteristics of EntryStore listed, see also paper 6 and (Hannes Ebner & Palmér, 2012):

• **Use of named graphs** - an entry (formerly referred to as SCAM record) was captured as three things, a resource, a Named Graph that contained metadata about the resource, and the administrative information of the entry, also expressed as a Named Graph. The administrative information contained access control, provenance and the characteristics of the entry.

• **Five type schemes** - the type schemes were identified to capture the various aspects of entries. LocationType was used to express if the resource and metadata were managed locally. RepresentationType was used to distinguish between resources that had digital representations and those that were only providing identities for physical objects, abstract ideas etc. BuiltinType was used to distinguish between specially treated resources, such as folders, and general resources that the system had no special knowledge about. As always, the well-known mimetype was used to describe the format of the resource. Finally, an ApplicationType was reserved for allowing the Applications built on top of EntryStore to define their own types, maybe to trigger special behavior.

• **RESTful API** - EntryStore was extended to provide access via a RESTful API, where resource, metadata, and administrative information was made accessible on separate URIs.

• **SPARQL and Apache Solr** - a SPARQL endpoint was introduced to allow searching against the triple space. The Apache Solr library was included to provide efficient search against a set of common metadata fields, as well as against the URIs used and the types of the entry.

**Validation**

EntryStore was first used in the Organic.Edunet project (H. Ebner et al., 2009) where the aim was to provide educational material on organic agriculture in Europe. Its first purpose was to serve as an aggregator of Learning Object metadata from several other (non-semantic web) repositories. Hence, when the metadata was harvested, it was transformed into RDF and exposed as entries with remote metadata. The second purpose was to allow content experts to go through the material and enhance it with additional metadata fields, most specifically using an ontology for organic agriculture. EntryStore also served as a repository where content authors described

---


70 Under the name of Confolio, see [http://oe.confolio.org](http://oe.confolio.org)
and sometimes uploaded new content. Finally, EntryStore was harvested by a portal that provided various forms of search-based access to the content, including advanced semantic search.

In the Hematology Net project (H-net) EntryStore was used\textsuperscript{71} to handle competency descriptions of hematology experts in Europe (mostly medical doctors). This was achieved via an expanded personal profile containing references to a range of topics paired with a competency level. The EntryStore also contained educational resources that could be tagged with the same (competency) topics and also a corresponding topic-based search.

Both Organic.Edunet and H-net used the EntryScape application for managing the content. But experiments have been performed with other applications. For instance, a smaller OpenSocial Gadget has been developed that exposes a simple folder view of content.

**Observations**

By exchanging the dependency from the Jena triplestore to Sesame\textsuperscript{72}, a range of different RDF storage solutions became available. This is due to the Sesame SAIL API (Storage And Inference Layer), which is supported by several third-party high-performance RDF stores, such as Bigdata and Virtuoso.

Since SPARQL has no knowledge of access control, a separate repository containing only public metadata was introduced to allow queries to be executed without risking to expose protected data. The drawback with this approach is that protected non-public data that the user is authorized to see will not show up in the results. The only feasible alternative is to rewrite queries on the fly to take into account that all triples matched in a query must reside in Named Graphs that the user had authorization to. However, this approach has several problems. First, it requires that the access control rules can be expressed in a SPARQL query, second the resulting queries would not be as efficient. Since the added benefits of this approach was considered smaller than the cost of development and expected poor results regarding efficiency, it was not considered worth the extra effort. Instead, the focus shifted to enhance the Solr index so that most queries could be answered there.

The development in SCAM 2 and SCAM 3 towards a platform, where a range of different applications can be built on top of a common service, culminated with EntryStore. In fact, EntryStore does not contain a single line of application-specific code. The only remaining reason for deploying EntryScape on the same server as EntryStore is the lack of support for cross-site scripting in the browsers that are in use today. With better support for CORS\textsuperscript{73}, this limitation will slowly fade away. If needed there are workarounds to overcome this limitation in a shorter perspective. Note that this is not a problem when developing RESTful AJAX native applications, for instance on mobile platforms.

The SPARQL Graph Store HTTP Protocol, (Ogbuji, 2012), resembles part of the RESTful API offered by EntryStore and should be supported in a future version.

---

\textsuperscript{71} Under the name Confolio, see http://hematology.confolio.org
\textsuperscript{72} http://openrdf.org
\textsuperscript{73} CORS stands for Cross-Origin Resource Sharing, see W3C draft at: http://www.w3.org/TR/cors/
6.4 Summary

From the start the four iterations of development have addressed the second obstacle as well as the derived requirements. Even though the requirements have been more or less fulfilled from the start, the technical solutions applied in each iteration have had their strengths and weaknesses. Most of the strengths have been successfully carried along to later iterations where also many of the weaknesses have been successfully addressed. The most important features of the latest iteration, EntryStore, are:

- Reuse an existing triplestore (in fact a quad store)  
  \textit{Introduced in iteration 2, improved in iteration 4.}

- Manage RDF on the level of entries and contexts.  
  \textit{Introduced in iteration 4, improves upon work in iteration 1 and 2.}

- Describe the character of an entry via five orthogonal type-schemes.  
  \textit{Introduced in iteration 4, improves upon work in iteration 1 and 2.}

- Manage a resource and its metadata in a single entry, containing related RDF, provenance, access control and other control information.  
  \textit{Introduced in iteration 4, improves upon work in iteration 2 and 4.}

- Support graph-based (SPARQL) and text-based (Solr) search.  
  \textit{Introduced in iteration 4, improves upon work in iteration 2.}

- Support backup- and restore functionality.  
  \textit{Introduced in iteration 2.}

- Offer a RESTful API, Linked data support and SPARQL-endpoints  
  \textit{Introduced in iteration 4.}

The known weaknesses of the last iteration are:

1. The RESTful API is yet not aligned with existing initiatives, such as SPARQL Graph Store HTTP Protocol, (Ogbuji, 2012), Atom Publishing Protocol, or Leap2R.

2. Support for inference is still lacking.

3. Support for migrating between vocabularies is lacking.

4. SPARQL queries only work against public data.

5. There is no version control of Semantic Web data.

It should be noted that weakness one, two and three can and will be addressed as soon as possible. Weakness four and five are more problematic since there are technical challenges that need to be resolved regarding access control in SPARQL queries, as well as how to uniquely address parts of RDF graphs.
7. Recommendations

This chapter will focus on recommendations for how to build learning applications based on Semantic Web technology. This corresponds to the third obstacle identified in section 4.3:

_Lack of recommendations for how to build learning applications based on Semantic Web technology_

The recommendations are based on experience from practical development of learning applications, including the iterations described in section 5.5 and 6.3. They are also the result of theoretical work reported in the papers, especially paper 1, 5, and 6. The relations derived in chapter 3 between architectures, technologies and application types have been crucial in finding the appropriate formulations of the recommendations.

The recommendations are targeted towards learning applications by having a focus on flexibility of a wide range of different knowledge expressions, both subjective and objective, while, at the same time considering how to best support communication and collaboration around them. If the focus would have been on building applications for a more specific domain, perhaps targeting a specific user group, the recommendations would almost certainly have been different. For instance, it is likely that the recommendations would have had a higher focus on how to work with ontologies, support inference, and compatibility with legacy systems.

Note that the recommendations deviate from the e-learning framework described in paper 1. The main reason is that at the time of writing, technologies such as Semantic Web and Web Services were still young, and the suggested e-learning framework was more based on theory and expectations on how the technologies would pan out, rather than on practical experience.

7.1 Rely on the Web Architecture

You should always rely on the Web Architecture when building learning applications. Specifically, if there is a need to provide stable identifiers, use URIs, if there is a need to transport data, use established web protocols, preferably HTTP. Furthermore, it is recommended that your offering is discoverable on the web, so that it can be found via search engines, linked to from web pages, recommended in social networks etc.
But perhaps most importantly, the Web Architecture is a requirement for Semantic Web technologies, specifically exposing learning data as resources which can be referenced and turned into targets for statements via their URIs.

7.2 Commit to REST and get guidance from ROA

Build your learning applications using the REST architectural style and lean on the best practices of ROA when more concrete advice is needed. REST and ROA carry with them a deeper integration with the web architecture, especially when working with resources and resource representations that incorporate links to other resources.

There are many added values of REST and ROA, for instance simplicity and scalability. From the Semantic Web perspective the focus on resources and their representations is perhaps the most important. Since REST prescribes a quite sparse uniform interface, that is, the methods of HTTP (GET, PUT, POST, DELETE), the semantics is concentrated to the resources and how they are related. The focus on relations between resources matches the declarative character of RDF (stating facts on resources). In comparison, Web Services has a procedural character (invoking methods at a distance), which cannot be captured directly in RDF without first introducing a new vocabulary for services.

7.3 Expose your information as Linked Data

In addition to REST and ROA, use linked data in the form of RDF to express your structured data, instead of inventing your own data expressions. Also connect your data by referring to resources in other linked data datasets. These connections will strengthen the value of your data and, if your data is openly available, will increase its chance of being reused in new settings.

As linked data is expressed in RDF, there is a range of formats to choose from. In light of the recommendation below on "RESTful AJAX Web Applications", a lightweight format suitable for consumption in JavaScript clients is preferable, such as JSON-LD or RDF/JSON.

To maximize interoperability it is encouraged to reuse existing terms (element and value vocabularies) whenever possible. Only invent new terms when what you need to say is different or more specific than what can be expressed with existing terms. Remember that URIs should not be visible to the end-users, hence, consistency in choice of terms (e.g. similar URI namespaces) is not a reason for introducing your own terms.

7.4 Base your application on a read/write RDF framework

Use an existing framework for handling RDF if your learning application needs anything like access-control, manage resources together with their metadata, keep track of provenance, free-text search in addition to a SPARQL endpoint etc. The framework you choose should, as a minimum, provide a RESTful API that allows CRUD operations (Create, Retrieve, Update, and Delete) for managing RDF as linked data.

The EntryStore framework has served the purpose for the learning applications mentioned in this thesis, but there are other candidates. Note that a triplestore such as Sesame is not enough, since it does not provide a RESTful API for manipulating RDF as linked data.
7.5  Build Web Applications first

You should offer your learning application as a web application if possible. This lowers the threshold for learners to start using your application. It also allows you to make your application cross-platform from the start, ranging from traditional desktop environments to mobile platforms.

Another reason for choosing web applications is that the web has become the natural place to seek information, communicate, and collaborate. As learning applications often include handling information, communication and collaboration, by analogy, it is natural for users to expect a web-based solution for learning applications as well.

7.6  Build RESTful Ajax Web Applications

When building web-based learning applications use the RESTful Ajax approach rather than the static web-page, progressive enhancement or RPC Ajax approaches. In practice this means that the web application should be loaded as an initial HTML web-page with associated CSS and a JavaScript. Later, in response to user interactions or initiated Ajax requests, the application can change the user interface via D-HTML techniques. Naturally, the application should minimize page reloads and do all asynchronous communication with the server via RESTful calls.

A clear benefit of relying on the RESTful Ajax approach is that the application can work directly with the Semantic Web data. There is no need to introduce a new API or new data expressions which would be the case if the learning application relied on the RPC Ajax approach. The RESTful Ajax approach also has the benefit of allowing the learning application to load Semantic Web data from other sources with the same basic understanding of data.

7.7  Use a framework to help you present and edit RDF

Use a framework for editing and presenting RDF if your learning application needs to present arbitrary RDF, edit non-trivial RDF, or needs to be flexible with respect to what to edit. In general, to support changing requirements, the framework should not be tied to a specific standard. Instead it should rely on a configurable mechanism for expressing which element and value vocabularies to use. Such a framework is especially useful when the developers of the application are not used to work with Semantic Web technologies, or when they are not experts on which vocabularies to use in each situation.

The RForms framework (and its predecessors) have served these purposes for the learning applications mentioned in this thesis. To the authors knowledge at the time of writing, there are no other independent frameworks\(^{74}\) that allow editing and presentation of RDF in a configurable manner.

\(^{74}\) The SAHA metadata editor is close, but it is restricted to work with a specific solution for storing RDF.
8. Conclusions

Work that would lead up to this thesis began in 2001, the same year when Tim Berners-Lee coined the term Semantic Web (Berners-Lee et al., 2001). At this time the web was beginning to mature, with capable web browsers and relatively stable specifications of the standards involved. But no one had heard of concepts like Web2.0 or Ajax, and very few had tried to write a blog or join a social network.

Over the last decade computational technologies have changed dramatically, which is most prominently visible in how the web has evolved. There have been many forgotten debates over competing technologies, although some refuse to be settled, such as the SOA versus REST debate. Within the area of Semantic Web there is another debate between those that advocate the use of ontologies and those that focus more on the web part of Semantic Web, typically arguing for the approach of Linked Data.

From the discussions, especially in chapter 3, it is evident that the author has been influenced by these developments, and from the perspective of Learning Applications based on Semantic Web technology has found arguments to take part in some of these debates. Some of the results are described in chapter 7 - in the form of recommendations.

Regarding the overall structure, this thesis started with a vision of how Semantic Web technology could provide a basis for better learning applications, with respect to expressibility and quality of conversation on a growing range of topics. To make the vision more credible, in chapter 2 the thesis has discussed the flexible nature of RDF and how this can benefit learning.

In Chapter 3 architectures and technologies relevant to Semantic Web were thoroughly discussed. Since Semantic Web technologies are firmly based in the Web Architecture, it was argued that the Web Architecture is a good point of reference to which all other architectures and technologies can be compared. The comparison focused on how architectures utilize and integrate with the Web Architecture, where utilize means relying on the web architecture to transport data, while integrate means using the concept of resources and URIs internally in order to make the architecture or technology become part of the web. The main result of this discussion was that Web Services do not integrate with the Web Architecture, and the implication of this important fact is that Web Services are less suited - than for example REST-based approaches such as Linked Data - to be used in combination with Semantic Web technologies.
Chapter 3 also introduced seven categories of applications, called application types, and discussed how they relate to the architectures and technologies. The final result of this discussion is a derivation of the relationships between architectures, technologies and application types. The most clear result is that RPC-based application types are less useful in combination with Semantic Web technology, since they are related to Web Services. The conclusions of the derivation are drawn as part of the recommendations presented as a response to obstacle 3.

8.1 Research questions

The main goal of the thesis - to provide guidance for development of learning applications based on Semantic Web technologies - led to two research questions:

I. What are the main obstacles when building Learning applications based on Semantic Web technologies 

II. How can these obstacles be overcome using state-of-the-art web technologies and platforms

The first research question was addressed in chapter 4 where three obstacles were identified based on problems encountered during development of the two learning applications EntryScape and Conzilla. The second research question was addressed in chapters 5-7, that is, one chapter per obstacle. The text below shortly summarizes how the obstacles have been addressed - and to some extent overcome:

Obstacle 1: Lack of non-expert and user friendly solutions for presenting and editing Semantic Web data that is not hard-coded to use a specific vocabulary.

The solution introduced in chapter 5 is an independent configurable framework for presenting and editing RDF. The framework has gone through 6 iterations of development, and the last iteration is called RForms. The major implemented features of the framework were shortly summarized in section 5.6 as:

1. A configuration mechanism that is easy to understand and work with.
2. Configurations can be assembled from smaller dedicated configurations, sometimes referred to as formlets.
3. Support for match-all mode, where many configurations, formlets, are combined to match as much as possible of a given RDF graph.
4. Model and view separation, which makes it possible to have multiple presentation- and editing views.
5. RDF expressions are always minimal and correct according to the configuration.
6. A stand-alone JavaScript library that is easy to integrate in web applications, taking only an RDF graph and a configuration as input.

For the learning applications Conzilla and EntryScape, the framework has proved to be useful and has helped to overcome the obstacle, as discussed in section 4.3. Even though this is no guarantee that the framework will be of use in other settings, it is recommended to consider the six features above when looking for presentation and editing solutions for RDF.
**Obstacle 2: Lack of solutions that can create new, and manage existing, resources and related Semantic Web data, both privately and in collaborative settings.**

This obstacle is addressed in section 6 by breaking it down into five requirements: manage any RDF, manage chunks of RDF, manage web linking, manage resources, and manage private and shared information. The solution presented, EntryStore, meets the requirements more or less by design. Furthermore, as discussed in section 6.4, the following features of EntryStore are considered to be the most important:

- Reuse an existing triplestore (in fact a quad store)
- Manage RDF on the level of entries and contexts.
- Describe the character of an entry via five orthogonal type-schemes.
- Manage a resource and its metadata in a single entry, containing related RDF, provenance, access control and other control information.
- Support graph-based (SPARQL) and text-based (Solr) search.
- Support backup- and restore functionality.
- Offer a RESTful API, Linked data support and SPARQL-endpoints

The portfolio system EntryScape, a RESTful Ajax web application, is built upon the generic services offered by EntryStore, proving its value. The Conzilla application does not yet use EntryStore, but future versions will probably do so, since this would solve a few known problems. Furthermore, when building learning applications based on Semantic Web technologies - even if EntryStore is not chosen as a basis for development - the requirements and features discovered may provide useful guidance.

**Obstacle 3: Lack of recommendations for how to build learning applications based on Semantic Web technology**

In chapter 3 a range of relevant architectures and technologies were analyzed. The main outcome was a derivation - see figure 8 - of how architectures and technologies support each other and various application types. This derivation, together with the experience gained during the iterative development accounted for in this thesis, have led to seven recommendations described in chapter 7. The recommendations are:

1. Rely on the Web Architecture
2. Commit to REST and get guidance from ROA
3. Expose your information as Linked Data
4. Base your application on a read/write RDF framework
5. Build Web Applications first
6. Build RESTful Ajax Web Applications
7. Use a framework to help you present and edit RDF

This thesis acknowledges that when it comes to choosing an appropriate technology for a specific project there is no absolute truth, there are simply so many factors involved, not the least the pre-existing knowledge of the developers involved. Nevertheless, this thesis does try to provides a
8. Conclusions

well thought through rationale for why these recommendations are given. They also fit well together, as is shown by both practical development and the more theoretical derivation in chapter 3.

8.2 Contributions of this thesis

The major theoretical contributions of this thesis are:

**Seven application types - section 3.7**

**Derivation of standards, technologies & application types - section 3.8**

**Three obstacles for building Semantic Web based learning applications - section 4.3**

**Five categorizations for solutions to edit/present RDF - section 5.1-5.4**

**Five requirements on solutions for managing resources and RDF - section 6.1**

**Seven recommendations for building Semantic Web based learning applications - chapter 7**

As regards the practical development that has formed the background of this thesis, the author believes that both SHAME/RForms and EntryStore/EntryScape are sound frameworks that can be of use to a wider community. This has already been demonstrated in the case of EntryScape, as evidenced by the Organic.Edunet and the H-net projects.

8.3 Future Work

It is evident that there is a lot of future work with regard to the learning perspectives presented (and not presented) here, and how these perspectives can be more effectively and efficiently supported by technology.

However, as pointed out in chapter 2, the focus of this thesis is technical. Hence, we will now go through a few open issues of technical character that the author considers to be of relevance for the topic of this thesis, as well as a few anticipated further developments of the frameworks described above. Although not demonstrated in this thesis, it can be expected that some of these issues are of importance for enhancing the conversational quality of the learning process.

**Editable Linked Data** - Perhaps one of the most important open issues is how to make linked data editable. The SPARQL 1.1 Graph Store HTTP Protocol, (Ogbuji, 2012), seems very promising, but it just provides part of the solution, since it lacks access control and provenance. The recently launched Linked Data Platform Working Group\(^\text{75}\) is an interesting forum for raising these and other issues that have been encountered during development of SCAM/EntryStore, as well as the discussion around overcoming obstacle 2.

**Model for entries** - Closely related is the issue of stabilizing the expressions used in EntryStore for controlling how entries are managed. In paper 6, the type schemes were introduced, shortly stating if a resource is an information resource or not, whether a resource is available on the web

\(^{75}\) [http://www.w3.org/2012/ldp](http://www.w3.org/2012/ldp)
or is an uploaded resource in EntryStore, whether there is external metadata etc. Preliminary work has started on an RDF Schema\textsuperscript{76}, but more work is needed, especially in order to better connect to other models such as the W3C provenance model\textsuperscript{77}.

**Hide that URI** - A less specific issue concerns how applications should avoid to show the URIs of links to resources when presenting Linked Data resources. Assuming that there is a label within the linked-to resource, how can the application get a hold of this label in an efficient manner. Pre-fetching all resources that are "one step away" is certainly possible, but for RESTful Ajax Web applications this is risky, since it may result in too many requests, affecting both reliability and speed. An alternative would be to establish the principle that services that offer linked data should regularly cache all resources that are linked to from a given resource. And when a resource is requested with a specific parameter, or HTTP header, the linked-to resources are included as well. A syntax like Trix or Trig\textsuperscript{78} could be used, where the original resource can be the default graph, and each linked-to resource can be a Named graph.

**Overlapping vocabularies** - One of the practices of Linked Data is to reuse and mix vocabularies in order to increase interoperability. But, what happens when multiple overlapping vocabularies exists? For example, the recent Schema.org\textsuperscript{79} initiative introduces new vocabulary that overlaps largely with many other vocabularies such as Dublin Core terms and FOAF. Is it the provider of Linked Data that is responsible for providing multiple vocabularies to maximize understanding for simpler consumers? Or is it the consumers' responsibility to be smarter and understand as many vocabularies as possible? Could a solution be that Linked Data providers use a single preferred vocabulary and only provide the additional vocabulary when a commonly agreed-upon parameter or HTTP header is provided, again using an appropriate format such as Trix or Trig. This would at least make updating Linked Data in a RESTful style more intuitive, since it would be possible to perform a PUT with what you received from a GET without polluting the data set.

**Generating RForms-templates** - Experiments have already been carried out to generate RForms-templates from RDF Schema and DSPs, see section 5.5.6. However, the results are seldom perfect, since additional information is missing, for instance, which fields that should be edited in text-areas with multiple lines, the order of presentation, and which kind of form-controls that should be used. When only RDF Schema is used (which is common), even more information is missing. Clearly it is possible to generate RForms-templates and then edit them manually, but if the vocabulary is updated regularly, such as for instance for the Schema.org vocabulary, this quickly becomes unmaintainable. In this case it might be useful to have another kind of configuration that only complements what can be found from RDF Schema or DSPs. This kind of complementary configuration format may be more important in the long run than the RForms-template itself. An interesting option would be to investigate if the DSP format can be extended or allow extensions to fill these gaps. However, the approach taken has to fit with the actual process, that is, it has to take into account who is going to provide the extra information. That is, if it is the same person that provides the RDF Schema, the DSP, or someone else altogether.

**Library of RForms-templates** - Another possibility would be to provide a service for managing a library of RForms-templates. For each vocabulary, a range of different RForms-templates could be maintained. The goal would be that such a service should be maintained by a community of experts, preferably people involved in standardization or responsible for maintaining the

\textsuperscript{76} http://code.google.com/p/entrystore/wiki/ReM3
\textsuperscript{77} http://www.w3.org/TR/prov-o/
\textsuperscript{78} http://www4.wiwiss.fu-berlin.de/bizer/TriG/
\textsuperscript{79} http://schema.org/
vocabularies. The added value would be simplicity for developers, making it easy to add support for various vocabularies. Perhaps presentation and editing interfaces can even be embedded directly from this service.

**HATEOAS for Linked data** - One of the architectural constraints of REST is that "hypertext as the engine of application state", often shortened as HATEOAS. This principle has far-reaching consequences regarding how to develop web applications, and raises quite a lot of controversy. For instance, there is a strong movement toward building RESTful web services using custom JSON-based data-structures that cannot be used without out-of-band knowledge\(^\text{80}\). That is, not all knowledge is encoded in the response in order for a generic client to know how to proceed. A combination of Linked Data formats, such as JSON-LD or RDF/JSON, with site-wide instructions for how to work with resources such as WADL\(^\text{81}\) or the Web Host Metadata\(^\text{82}\) RFC, may provide a way forward. Also, via content-negotiation, the Linked Data format can quite simply be embedded into a HTML page that bootstraps into a web application via a standardized JavaScript library that understands the Web Host Metadata or WADL. Such a generic web application would not be very user friendly, but it would provide users with interaction possibilities and fulfill the HATEOAS requirement, without sacrificing the usefulness of the data-structures when constructing more specific RESTful Ajax web applications. This approach needs further elaboration, development and testing in order to verify that it can be said to fulfill the HATEOAS requirement. It is perhaps even more important that there is added value, for instance simplified testing, for developers that today create RESTful services without the HATEOAS constraint being fulfilled.

**Future of semantics** - Recent developments indicate that Semantic Web technologies are slowly becoming more mainstream among developers. The most prominent examples are the industry-backing of the Schema.org initiative, and how the so-called knowledge graph helps to provide better search results in Google searches. But how will this affect the use of vocabularies and web application development? Will the requirement to show up in the knowledge graph overshadow the need for more precise expression? That is, will it be more important to chose a vocabulary that is understood by search engines than to chose a vocabulary that better expresses what you want to say? Will search engines force us to use RDFa (a way to embed RDF statements in HTML) in progressively enhanced web applications, or will they understand Linked Data directly? Will they perform content-negotiation to get the Linked data version if both RDFa and Linked Data exist in parallel?

**Improved recommendations** - The current recommendations for building learning applications based on Semantic Web technology will hopefully provide a good start as a common base for a wide range of platforms. But more specific recommendations could be provided for learning applications if they are targeted towards mobile platforms, the widget format, browser plugins, appearance in application stores etc. The recommendations can probably also be made more specific if open issues as those listed above are resolved.

---

80 The opposite, in-band knowledge means that the knowledge on how to use the data-structures is encoded in the messages, typically via a combination of understanding HTTP and a format (indicated by the Internet media type) that conveys information on which operations that are allowed and on which resources.
81 [http://www.w3.org/Submission/wadl/](http://www.w3.org/Submission/wadl/)
9. Summary of Papers

The following sections provide a summary of each paper, including bibliographic information.

9.1 Paper 1: E-Learning in the Semantic Age

**Year:** 2001  
**Authors:** Matthias Palmer, Ambjörn Naeve, Mikael Nilsson  
**Published:** Proceedings of the 2nd European Web-based Learning Environments Conference (WBLE 2001), Lund, Sweden, October 24-26, 2001.  
**Contribution:** The author of this thesis wrote the paper with input and corrections from the second and third authors.

The paper provides an appropriate starting point for the thesis, since it gives an overview of the main learning technologies at the time of writing, outlines a few problems ahead, and suggests a general direction of research. Hence, the paper is more of a position paper with a visionary and sometimes argumentative character than a standard, result-oriented research paper.

The paper discusses a few of the current standards and initiatives in the learning technology domain, such as SCORM and MIT OKI, notably how they are intended to be used in LMS:es. Several problems with LMS:es are identified, largely concerning the lack of flexibility for supporting new pedagogical approaches and learning tools. These problems center on how to use metadata in a collaborative setting, for example in comments, ratings, and reviews.

As an alternative solution, a learning framework consisting of five layers is introduced: a transport layer, an exchange layer, a semantic layer, a service layer, and an application layer. There is also a framework control layer, where services can be registered using for instance WSDL. Since the learning framework would be based on the principles of the Semantic Web, it would be more open-ended than comparable LMS:es that often use a few fixed standards. The paper then describes some existing applications such as Conzilla, Edutella and the Virtual Workspace Environment (VWE) and discussed how they fit into the learning framework.

In retrospect, the paper is visionary with respect to how Semantic Web
technology can be used for learning. However, the learning framework introduced here has not been pursued further. One reason is the suitability of relying on Web Services in combination with Semantic Web technology. Another reason is that the suggested framework would depend on a few centralized solutions which is problematic since there exist no central authority today that could take on the responsibility of maintaining those solutions.


**Year:** 2002  
**Authors:** Mikael Nilsson, Matthias Palmer, Ambjörn Naeve  
**Published:** Proceedings of the 11th World Wide Web Conference (WWW2002), Hawaii, USA, 2002.  
**Contribution:** The author of this thesis was mainly responsible for the introduction, background and requirements chapters while the second author was mainly responsible for the design and implementation chapters.

The first part of the paper discusses some common misconceptions regarding the nature of metadata. The second part focuses on how to create, publish and retrieve metadata. It is argued that non-authoritative metadata, such as comments, ratings, etc. will often be created in small inter-connected chunks and will therefore require more flexible editors. Different approaches for storing metadata are also discussed in the, touching on central server approaches and contrasting with how it could work in p2p networks.

The paper continues in a more practical manner by discussing a few concrete tools such as the p2p network Edutella, the conceptual browser Conzilla and the digital portfolio system SCAM portfolio\(^{83}\) for managing content and describe it with metadata. Finally, the paper introduces a learning scenario where these tools are used in a novel way.

The discussions regarding the nature of metadata in this paper are still valid and important. The ideas regarding the role of digital portfolios and how a concept browser can be of use are also still valid. However, the envisioned p2p infrastructure Edutella, for managing educational metadata has not been realized. The Edutella infrastructure gained a lot of attention but there was little need for this kind of solution. For instance, Edutella relied on the assumptions that there would be a multitude of small metadata providers that frequently would connect and disconnect from the p2p network. However, such intermittently connected providers have still not materialized, at least not until the time of writing of this thesis. Moreover, there were also unresolved problems regarding for instance how to route queries to parts of the network that contained the most relevant metadata.

---

83 SCAM portfolio was later renamed to Confolio and then to EntryScape.
9.3 Paper 3: The SCAM framework - helping semantic web applications to store and access metadata

**Year:** 2004  
**Authors:** Matthias Palmer, Ambjörn Naeve, Fredrik Paulsson  
**Contribution:** The author of this thesis wrote the paper with input and corrections from the second and third authors.

The paper introduces SCAM\(^\text{84}\), a metadata management system. The main contribution of the paper is that it introduces two levels of granularities for managing metadata: SCAM records and SCAM contexts. A SCAM context consists of a set of SCAM records that are advantageously administrated together, for instance all records that are controlled by a single user by an organizational body. A SCAM record is defined via a graph algorithm called anonymous closure that starts from a resource in an RDF graph and extracts a suitable sub-graph. This definition makes record independent of any specific metadata schema, such as Dublin Core or LOM, while at the same time capturing a relevant metadata record of a single resource. Such a record could for example describe a news item, a learning resource, a person etc. In addition, SCAM provides access-control on both SCAM records and SCAM contexts.

In parallel to the access-control solution, SCAM also provides generic metametadata, where information such as type of resource, date of creation and modification is stored. The metametadata and the access-control are expressed as small graphs inside of each SCAM record. This is practical but the semantics is somewhat dubious, an issue which is discussed in the paper.

SCAM is built as a J2EE application on top of the JBoss platform, and it uses the Jena triplestore as back-end. There is also a middleware layer, which provides useful building blocks for constructing a variety of applications on top of SCAM. At the time when the paper was written, there was approximately 10 different applications built using SCAM. One of them was the SCAM portfolio, which also relies on the SHAME metadata editor framework.

The paper presents solutions that could bridge the gap between what pure triple stores - such as Jena, Redland, Sesame etc. - offered at the time of writing, and what applications - such as digital portfolios - would need for large-scale and many-user metadata management. Note that Named Graphs was not invented at the time the paper was written. See paper 6 for how Named Graphs and other ideas have been incorporated in SCAM version 4. However, the basic idea of having two levels of granularity for managing RDF is still relevant.

9.4 Paper 4: Conzilla – a Conceptual Interface to the Semantic Web

**Year:** 2005  
**Authors:** Matthias Palmer, Ambjörn Naeve  
**Published:** Invited paper at the 13:th International Conference on Conceptual

---

\(^\text{84}\) Since the time of writing, the SCAM acronym has been changed to mean Standardized Contextualized Access to Metadata. Later it has been renamed EntryStore.
This paper describes the second version of the tool Conzilla that aims to support building and presenting networked knowledge structures that can enhance the learning process in various ways. Conzilla is an implementation of a Concept Browser. It creates overviews of concepts and concept-relations in what is referred to as Context-maps. A Context-map can be expressed using different graph-based diagram styles. As a concept browser, Conzilla also supports the association of concepts (or concept-relations) with content-components, which is most often used to provide concrete examples of what a concept (or concept-relation) stands for. Content-components, concepts, concept-relations, as well as the Context-maps themselves, are described with metadata according to established standards. In order not to clutter the view of the Context-map, such metadata is not permanently exposed. Instead, metadata is exposed upon request in a separate form-like interface, which is generated via the configurable metadata-editing framework SHAME, see paper 5.

An important feature of a Concept Browser that distinguishes it from many other initiatives is the support for navigation between Context-maps. The navigation can be mediated explicitly via hyperlinks on concepts and concept-relations. It is also possible to navigate between two Context-maps whenever at least one concept has been reused and appears in both maps.

The graph-based nature of Context-maps, the need for public identifiers in order to allow reuse, and the need for metadata expressions using established standards are all strong arguments for using RDF. The paper shows how they have influenced the second version of Conzilla, which has switched to using RDF for representing Context-maps. The paper also discusses the implications of this switch on the design of Conzilla.

One of the major points of the paper is the division of the RDF expression into three layers concerned with respectively information, presentation, and style. The presentation layer of a Context map contains layout information of the map. The style layer provides information on how to style concepts and concept-relations based on their types. Finally, the information layer contains representations of the actual concepts, concept-relations, and content-components. More specifically, any metadata that provides information that is not part of the graphical presentation belongs to the information layer. It is the information layer that is most likely to be understood by other (non-Concept-browser) applications. And conversely, existing information expressed in RDF can be complemented in Conzilla with layout and style to be visualized as concepts, concept-relations and content-components within one or several Context-maps.

The paper discusses at some length how references between the presentation and the information layer becomes problematic when they are not expressed in the same RDF graph. The paper also mentions that Conzilla could benefit from integration with an RDF-based storage-and-access solution such as SCAM. Even though this discussion exceeds the scope of this thesis, the design of version 4 of SCAM described in paper 6 is intended to simplify such integration.

Finally, the paper presents a comparison with other tools, both from the perspective of conceptual browsing and from the perspective of RDF editing. Both perspectives highlights features where Conzilla was unique (at the time of writing).
9.5 Paper 5: Annotation profiles: Configuring forms to edit RDF

**Year:** 2007  
**Authors:** Matthias Palmer, Fredrik Enoksson, Mikael Nilsson, Ambjörn Naeve  
**Published:** Proceedings of the International Conference on Dublin Core and Metadata Applications, Singapore 28 - 31 August 2007  
**Contribution:** The author of this thesis was mainly responsible for the introduction, background and requirements chapters.

This paper introduces the Annotation Profile Model (AP model) as a configuration mechanism for metadata editors. In order to make effective use of the strength of this model in the editing process, three user roles are introduced. First, the AP-author describes new APs based on his or her expertise in metadata schemas/ontologies and various knowledge domains. Second, the AP-facilitator is responsible for selecting and making specific annotation profiles available, perhaps based on the characteristics of the tasks and the end users involved. Third, the AP-end-user edits the metadata. Note that the same person may take on more than one role.

The paper describes the AP model in detail, including its two fundamental constituents: the Graph Pattern and the Form Template. The Graph Pattern resembles a query language and is responsible for both capturing and creating sub-graphs of RDF triples. Existing RDF query languages, such as SPARQL and QEL, have been successfully used as syntax - albeit with a different semantics, which avoids the quite verbose expressions that would otherwise be needed. The Form Template provides a tree of form items that references the variables in the Graph Pattern. It also provides order, grouping, interaction hints, multilingual labels and descriptions, as well as hooks for external style sheets etc.

The paper also describes the process where an Annotation Profile is used to match RDF data into an form. This chain of instructions includes three intermediate steps. First, the graph pattern is matched against the RDF data, which yields a hierarchy of Variable Bindings. Second, the variable bindings are used to instantiate the Form Template into a Form Model, where parts of the Form Template are duplicated or left out, depending on the amount of matched RDF data. Third, the Form Model is used to generate an actual graphical user interface.

The paper concludes by pointing out that the SHAME framework is written in Java and includes an application front-end based on the Swing framework and a web front-end that is generated via the server-side template language named Velocity. At the time of writing of the paper, another implementation based on RESTful principles and other Web2.0 techniques was foreseen. It has since been realized within a successor to SHAME called RForms.


**Year:** 2008  
**Authors:** Hannes Ebner, Matthias Palmer  
**Published:** Proceedings of the First International Workshop on Mashup Personal Learning Environments (MUPPLE08), 388. 14-17, Maastricht, The Netherlands, September 2008.  
**Contribution:** The author has written the paper collaboratively with a special responsibility for the sections on RDF design and the Confolio use case.
The focus of the paper is on the fourth iteration of the SCAM framework (SCAM 4), which is a framework for managing resources and metadata together. A new design, with entries in contexts, is introduced. This design goes beyond what was possible with the version described in paper 3 - both with respect to the RDF expressions and the location of the resource and its metadata. There is also a focus on the usefulness of the framework from a mashup perspective, where application developers are able to reuse the framework and concentrate on user interface development.

The design is focused on three types. The representation type conveys whether or not the resource is an information resource. The built-in type is used to specify if the character of the resource is known to the framework. Examples are given by resources such as 'list', 'user', and 'group'. The location type, provides information about where the resource its corresponding metadata are located. For example, a location type of local means that both the resource and its metadata is maintained locally within the framework, while a location type of reference means that both the metadata and the resource are external to the framework.

The paper also describes an RDF expression which relies heavily on named graphs. A RESTful API that can access the information in the framework is also introduced. The paper ends with some details of an implementation (SCAM 4) and an application (Confolio) built on top of the framework.

The development of the SCAM framework did not stop when the paper was published. One of the main developments after the publication is the formalization into a model (expressed in RDF Schema), called the Resource and Metadata Management Model (or ReM³ for short).
References


Papers

The following six papers have all been published earlier with different requirements on layout and style. This information has been preserved in this thesis, and the papers will therefore not conform to the layout and style of the rest of the thesis. Only the page numbers have been changed so that each paper now starts with page number 1. Consequently, if you want to refer to a page in one of the papers you need to specify "page X in paper Y".