The Personal Reader: A Framework for Enabling Personalization Services on the Semantic Web

Nicola Henze, Marc Herrlich
ISI - KBS, University of Hannover
D-30167 Hannover, Germany
henze@kbs.uni-hannover.de, marcherrlich@gmx.net

Abstract

The *Personal Reader*[1] provides a framework for designing, implementing and maintaining web content readers, which provide personalized enrichment of web content for each individual user. The idea of the the Personal Reader is based on a rigorous approach for applying Semantic Web technologies: A modular framework of components / services - for visualizing the Personal Reader and providing the user interface, for mediating between user requests and available personalization services, and for providing personalized recommendations and access to web content forms the basis for the Personal Reader. In this paper, we describe the architectural outline of the framework as well as some implementation details, and discuss the approach with a reference - a "Personal Reader for Learning Resources".

1 Introduction

With the vision of the Semantic Web [Berners-Lee et al., 2001] in which machines shall be enabled to understand, process, and reason about the semantics of web resources, the question of personalizing the interaction with web content according to the individual needs of the end-user is at hand. The question on how to enable personalization functionality in the Semantic Web can be regarded from different viewpoints, involving on the one hand disciplines that investigate personalization functionality, e. g. data mining, machine learning, web graph analysis, collaborative approaches, adaptive hypermedia, and more. On the other hand, it evolves recent web technologies like e.g. ontological descriptions, reasoning and query languages for the (semantic) web, web services, etc.

In the Personal Reader project[2] we implement personalization functionality from the area of adaptive hypermedia and show how this adaptation functionality can be exposed as "Personalization Services" on the Semantic Web. The Personal Reader project provides a framework for designing, implementing and maintaining web content readers, which provide personalized enrichment of web content for the individual user. In this paper, we will present the main ideas on how RDF enables the communication, how learning resources and domain concepts are annotated for the Personal Reader, and which ontologies or standards are required and used for the Personal Reader.

[1]This work was partially supported by the Network of Excellence REWERSE (www.rewerse.net).

[2]www.personal-reader.de

We will describe the basic architecture of the Personal Reader framework (section 2), and discuss some implementation details of a "Personal Reader for Learning Resources" (section 3). At the end of the paper, we compare our approach to related work in this area and conclude with an outlook on current and future work.

2 Architecture

The Personal Reader features a distributed open architecture designed to be easily extensible. It utilizes standards like XML [XML, 2003], RDF [RDF, 2002], etc. and technologies like Java Server Pages (JSP)[Java Server Pages, 2004] and XML-based-RPC[XML-based RPC, 2004] to provide a personalization framework. The architecture is based on different web-services cooperating with each other to form a Personal Reader instance. Communication between these web-services takes places by exchanging RDF documents.

A Personal Reader instance consists of the following three main components:

- a Connector Service
- one or several Visualization/User Services
- one or several Personalization Service

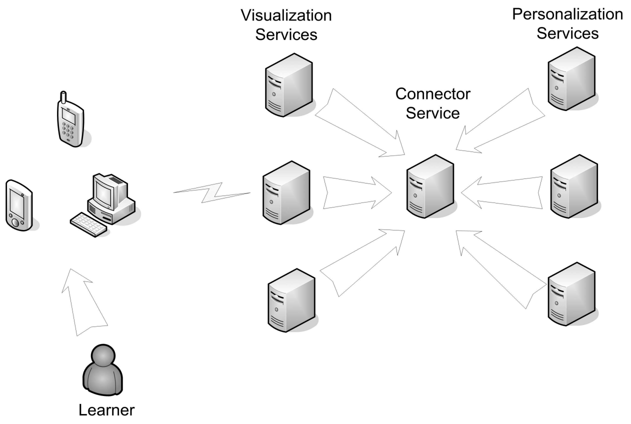


Figure 1: Architecture

One main goal of the Personal Reader architecture is to provide the user with the possibility to select services, which provide different or extended functionality, e.g. different Visualization or Personalization Services, and combine them into a Personal Reader instance.

Another key feature of the Personal Reader, in which it differs from many other personalization systems, is that the personalization processing is based on a learning resource description provided as an external RDF file. Therefore no modifications to the learning resources themselves have to be made. The Personal Reader tries to provide Plug & Play like behavior for adaptive personalization systems.

To achieve these goals several issues have to be solved:

- sufficient metadata descriptions for users and learning resources are needed
- mediation between the services and aggregation of the results have to be performed
- the results have to be presented to the user
- the actual reasoning/inference has to be done

The architecture of the Personal Reader is open in many ways. It is open in the meaning of:

- using open standards
- using open protocols
- being open to different implementations
- being open in the way it handles learning resources

The following subsections will describe in detail the metadata used by the different kinds of Personal Reader services and the services themselves.

2.1 Metadata

The Personal Reader needs metadata descriptions of users, courses, learning resources and ontologies. Additionally it uses metadata descriptions to perform the data exchange between the different services. All metadata used by the Personal Reader is based on well-defined RDF schema definitions and learning specific standards. This chapter explains in detail the metadata needed to describe users, courses, learning resources and ontologies. The interfaces between the services are explained in the chapters about the services themselves.

Describing users

Information about users is needed for the Personalization Service to be able to provide recommendations based on user interests, performance, goals and preferences. The user profile used by the Personal Reader is a RDF document, which consists of well-defined RDF properties and describes a learner's current learning state.

Here is an example of how an user profile may look like:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://.../rdf/l3s.rdf#" >
 <rdf:Description rdf:about="http://.../user#john">
  <rdf:type rdf:resource="http://.../rdf/l3s.rdf#User"/>
  <j.0:name>John Doe</j.0:name>
  <j.0:hasCourse>jtut</j.0:hasCourse>
  <j.0:hasVisited rdf:resource="http://.../index.html"/>
  <j.0:hasVisited rdf:resource="http://.../managingfiles.html"/>
  <j.0:done rdf:resource="http://.../managingfiles.html"/>
  <j.0:hasVisited rdf:resource="http://.../variables.html"/>
  <j.0:needAgain rdf:resource="http://.../variables.html"/>
 </rdf:Description>
</rdf:RDF>
```

User profiles are of the type `User` and feature the following properties:

- `name`
- `hasCourse`
- `hasVisited`
- `done`
- `needAgain`

Every user profile has to have exactly one `name` property, which gives the user's real name. Additionally it may have one or more of the other properties listed above.

`hasCourse` This property marks the courses an user is registered for. In the example above the user is registered for the course `jtut`. For every course there has to be one property of this type.

`hasVisited` This property marks learning resources an user has visited. Just like in the case of the `hasCourse` property there has to be one property of this type for every visited learning resource.

`done` This property marks learning resources the user has *explicitly* stated as done. This means the user thinks he has learned all the information provided by the resource and does not want the system to recommend it to him in the future. Again there has to be one property of this type for every learning resource marked.

`needAgain` This property marks learning resources the user has *explicitly* stated as to be needed again. This means the user thinks he has *not* learned all the information provided by the resource, yet and wants the system to recommend it to him in the future. `needAgain` is complementary to `done` as a resource can only be marked as done *or* as to be needed again. There has to be one property of this type for every learning resource marked.

Describing courses

Learning resources are usually aggregated into courses, e.g. a lecture or a tutorial about a specific topic. Of course the same learning resources may belong to several different courses. The Personal Reader needs information about courses and learning resources in the form of an external RDF description to do the course specific reasoning and inference. In addition to the information about the learning resources themselves a domain specific ontology is required to define relations between the different concepts of the learning material. As the Personal Reader is an open system the learning resource description as well as the ontology may both be not locally available but located for example on a webserver. Therefore a third document is needed to store the location of those descriptions. This document is called the course profile and it contains all the information the Personal Reader needs to know about a course in order to handle it properly. All those documents are based on well-defined schema definitions and standards like Dublin Core [Dublin Core, 2004] and LOM [LOM, 2002].

Here is an example of how a course profile may look like:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://.../rdf/l3s.rdf#" >
  <rdf:Description rdf:about="http://.../course#jtut">
    <rdf:type rdf:resource="http://.../rdf/l3s.rdf#Course"/>
    <j.0:name>Java Tutorial</j.0:name>
    <j.0:startPage>http://.../index.html</j.0:startPage>
    <j.0:location>http://.../sun_java_tutorial.rdf</j.0:location>
    <j.0:ontology>http://.../java_ontology.rdf</j.0:ontology>
  </rdf:Description>
</rdf:RDF>
```

Course profiles are of the type `Course` and feature the following properties:

- `name`
- `startPage`
- `location`
- `ontology`

Like the user profile every course profile has to have exactly one `name` property, which stores a name or label for the course. This name is not necessarily needed to be unique, it is merely intended for displaying the course in an user-friendly way. The courses are uniquely identified by a special URI. Additionally a course profile has to have exactly one of the other properties listed above.

`startPage` This property holds the URL of the page that is to be displayed as entry page to the learner. In most cases the root page of the course is given here but technically any other page is fine, too. It may even be a page that does not belong to the course at all but in that case of course the Personal Reader will not be able to calculate any recommendations. That is why normally the page given here should be a learning resource that is described somewhere in the RDF learning resource description.

`location` This property holds the URL of the XML/RDF file, which contains the learning resource descriptions. To be used within the Personal Reader this file must be accessible via the web. It is easily possible to have different descriptions for the same set of learning resources by creating a course profile for each description with different values for the `location` property.

`ontology` This property holds the URL of the XML/RDF file, which contains the ontology the learning resource description refers to. The rules that apply to this property are the same as for the `location` property described above.

Describing learning resources

Learning resources are described using Dublin Core and Learning Objects Metadata (LOM) standards. In addition to the resource description an ontology is needed, which defines the relations between the concepts used in the description. The ontology definition is based on RDF Schema and standard RDF.

Here is a short excerpt from an example description of the Sun Java Tutorial (a freely available online tutorial [Campione and Walrath, 2003]) showing how learning resources can be described for the Personal Reader:

```
<rdf:Description rdf:about="http://.../flow.html">
 <rdf:type rdf:resource="http://.../lom-educational#LO"/>
 <dc:title>Control Flow Statements</dc:title>
 <dc:subject rdf:resource="http://.../ontology.rdf#Looping"/>
 <dc:subject rdf:resource="http://.../ontology.rdf#Branching"/>
 <dcterms:isPartOf rdf:resource="http://.../index.html"/>
 <dcterms:hasPart>
  <rdf:Seq>
   <rdf:li rdf:resource="http://.../while.html"/>
   <rdf:li rdf:resource="http://.../for.html"/>
   <rdf:li rdf:resource="http://.../if.html"/>
   <rdf:li rdf:resource="http://.../switch.html"/>
   <rdf:li rdf:resource="http://.../exception.html"/>
   <rdf:li rdf:resource="http://.../branch.html"/>
   <rdf:li rdf:resource="http://.../flowsummary.html"/>
   <rdf:li rdf:resource="http://.../QandE/questions_flow.html"/>
  </rdf:Seq>
 </dcterms:hasPart>
</rdf:Description>
```

In the `type` property a definition from the LOM standard is used to mark the learning resource as a learning object and the `dc:title` property holds the title of this resource. The most important properties for describing learning resources are:

- `dc:subject`
- `dcterms:isPartOf`
- `dcterms:hasPart`

Of course for some personalization services used with the Personal Reader, there may be additional properties of interest but the ones listed above are the basic properties required.

There are two ways to describe relations between learning resources:

- relations defined using an ontology
- relations defined directly within the resource description

`dc:subject` Every learning resource may have one or more `dc:subject` properties referring to concepts defined in the ontology belonging to the course. In the example above the described learning resource refers to the concepts `Looping` and `Branching`, which are defined in a file called `ontology.rdf`. This is how relations between resources using an ontology can be defined.

`dcterms:isPartOf` This property directly defines a generalization relation to another learning resource. In this context generalization means, the related resource is a more general resource on the same topic.

`dcterms:hasPart` This property directly defines a detail relation to another learning resource, meaning the related resource is a more detailed resource on the same topic or an extension of the resource. In combination with `dcterms:isPartOf` this is how relations between resources can be directly defined within the resource description.

As already mentioned, the learning resource description refers to an ontology. Here is a short excerpt from the ontology for the Java programming language that is being used in the example above:

```
<rdfs:Class rdf:ID="Programming_Elements"/>

<rdfs:Class rdf:ID="Basic_Programming_Concepts">
  <rdfs:subClassOf rdf:resource="#Programming_Elements"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Control_Structures">
  <rdfs:subClassOf rdf:resource="#Basic_Programming_Concepts"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Branching">
  <rdfs:subClassOf rdf:resource="#Control_Structures"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Looping">
  <rdfs:subClassOf rdf:resource="#Control_Structures"/>
</rdfs:Class>

<rdfs:Class rdf:ID="if">
  <rdfs:subClassOf rdf:resource="#Branching"/>
</rdfs:Class>

<rdfs:Class rdf:ID="switch">
  <rdfs:subClassOf rdf:resource="#Branching"/>
</rdfs:Class>

<rdfs:Class rdf:ID="while">
  <rdfs:subClassOf rdf:resource="#Looping"/>
</rdfs:Class>

<rdfs:Class rdf:ID="do">
  <rdfs:subClassOf rdf:resource="#Looping"/>
</rdfs:Class>

<rdfs:Class rdf:ID="for">
  <rdfs:subClassOf rdf:resource="#Looping"/>
</rdfs:Class>
```

You can see how for example the concept `Branching` is related to super and sub concepts and according to this relations a learning resource referring to `Branching` is in the same way related to other learning resources referring to `Branching` or super or sub concepts of it.

2.2 The Connector Service

The Connector Service is the mediator between the Visualization Service and the Personalization Service or rather instances of these services. It is the key component of the Personal Reader architecture. The main task of the Connector Service is the conversion between the different formats of metadata descriptions used by Personalization Service instances and providing a generic interface for the Visualization Service instances. Besides this the Connector Service has to fetch the learning resource descriptions and the ontology from the web, aggregate all the documents and send them to the Personalization Service. Depending on the capabilities of the Personalization Service it may also be necessary to fetch a reasoning program, e.g. in the case of a rule-based Personalization Service in most cases the rules for processing a request are not implicitly known but have to be provided by the Connector Service together with the other documents.

These are the different steps performed by the Connector Service:

1. Parse a request from the Visualization Service.
2. Fetch additional data needed from the web (resource description, ontology, personalization program).
3. Choose a Personalization Service if there is more than one available.
4. Convert all data into the appropriate format for the chosen Personalization Service.
5. Aggregate all the data into one document.
6. Send the document to the Personalization Service.
7. Receive a result document from the Personalization Service.
8. Convert the results into the appropriate format for the Visualization Service and send them back.

It is important to stress, that these are the basic operations provided by every Connector Service implementation but of course a specific implementation may provide additional functionality like for example:

- caching mechanisms
- optimization strategies
- failsafe strategies

Defining an interface to the Visualization Service

The interface between the Connector and the Visualization Service is defined by several documents. It is important to distinguish between the actual interface to the service, which is defined by a Web Service Description Language (WSDL) specification and a Java interface, and the RDF documents that make up the other part of the interface. As the Java interface is by definition implementation depended, it is discussed in the chapter about the implementation. Here we define only the abstract interface.

The Connector Service expects from the Visualization Service:

- a pointer to the requested learning resource
- a pointer to the learning resource description
- a pointer to the ontology used
- the user profile

All of these documents were already discussed in the preceding chapters.

The Visualization Service expects from the Connector Service:

- a RDF document containing the results of the personalization process

Here is an example of how this document may look like:

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:j.0="http://.../rdf/l3s.rdf#"
 xmlns:dc="http://purl.org/dc/elements/1.1/">

<!--- part 1 --->
<rdf:Description rdf:about="http://.../flow.html">
 <dc:title xml:lang="en">Control Flow Statements</dc:title>
 <j.0:general rdf:nodeID="A0"/>
 <j.0:details rdf:nodeID="A2"/>
 <j.0:summary rdf:nodeID="A3"/>
 <j.0:quiz rdf:nodeID="A1"/>
</rdf:Description>

<rdf:Description rdf:nodeID="A0">
 <rdf:type rdf:resource="http://.../22-rdf-syntax-ns#Bag"/>
 <rdf:_1 rdf:resource="http://.../index.html"/>
</rdf:Description>

<rdf:Description rdf:nodeID="A2">
 <rdf:type rdf:resource="http://.../22-rdf-syntax-ns#Bag"/>
 <rdf:_1 rdf:resource="http://.../for.html"/>
 <rdf:_2 rdf:resource="http://.../while.html"/>
 <rdf:_3 rdf:resource="http://.../exception.html"/>
 <rdf:_4 rdf:resource="http://.../switch.html"/>
 <rdf:_5 rdf:resource="http://.../if.html"/>
 <rdf:_6 rdf:resource="http://.../branch.html"/>
</rdf:Description>

<rdf:Description rdf:nodeID="A3">
 <rdf:type rdf:resource="http://.../22-rdf-syntax-ns#Bag"/>
 <rdf:_1 rdf:resource="http://.../flowsummary.html"/>
</rdf:Description>

<rdf:Description rdf:nodeID="A1">
 <rdf:type rdf:resource="http://.../22-rdf-syntax-ns#Bag"/>
 <rdf:_1 rdf:resource="http://.../questions_flow.html"/>
</rdf:Description>

<!--- part 2 --->
<rdf:Description rdf:about="http://.../user#john">
 <j.0:recommended rdf:resource="http://.../while.html"/>
 <j.0:recommended rdf:resource="http://.../for.html"/>
 <j.0:recommended rdf:resource="http://.../branch.html"/>
 <j.0:recommended rdf:resource="http://.../if.html"/>
 <j.0:recommended rdf:resource="http://.../switch.html"/>
 <j.0:recommended rdf:resource="http://.../exception.html"/>
 <j.0:alreadyVisited rdf:resource="http://.../index.html"/>
 <j.0:alreadyVisited rdf:resource="http://.../flow.html"/>
</rdf:Description>

<!--- part 3 --->
<rdf:Description rdf:about="http://.../questions_flow.html">
 <dc:title xml:lang="en">
  Questions and Exercises: Control Flow
 </dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../while.html">
 <dc:title xml:lang="en">
  The while and do-while Statements
 </dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../if.html">
 <dc:title xml:lang="en">The if/else Statements</dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../index.html">
 <dc:title xml:lang="en">Language Basics</dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../switch.html">
 <dc:title xml:lang="en">The switch Statement</dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../for.html">
 <dc:title xml:lang="en">The for Statement</dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../flowsummary.html">
 <dc:title xml:lang="en">
  Summary of Control Flow Statements
 </dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../branch.html">
 <dc:title xml:lang="en">Branching Statements</dc:title>
</rdf:Description>

<rdf:Description rdf:about="http://.../exception.html">
 <dc:title xml:lang="en">Exception Handling Statements</dc:title>
</rdf:Description>

</rdf:RDF>
```

The response document can be divided into several parts:

- a part containing resource specific annotations (here for `flow.html`)
- a part containing user specific annotations (here for

the user `john`)

- a part containing additional information needed, e.g. titles for displaying the learning resources

The first part contains the learning resource specific annotations using well-defined properties in conjunction with RDF containers. Each property refers to a container of the type `Bag`, which contains a collection of resources matching that property.

`general` This property marks resources, which are generalizations of the referred learning resource. Meaning these resources are more general on the same topic than the described resource.

`details` This property marks resources, which are details of the referred learning resource. Meaning these resource are more detailed on the same topic than the described resource.

`summary` This property marks resources, which provide a summary for the topic the referred learning resource belongs to.

`quiz` This property marks resources, which provide a quiz or exercise for the topic the referred learning resource belongs to.

The second part contains the user specific annotations. In the example document above only the `recommended` and the `alreadyVisited` properties are used but in general it may contain a third property called `notRecommended`.

`recommended` This property marks learning resources, which are recommended for the user to look at taking into account his current learning state.

`notRecommended` This property marks learning resources, which are not recommended for the user, e.g. because they have preferences like other learning resources the user has not learned, yet.

`alreadyVisited` This property marks learning resources the user has already visited at least once.

The third part contains additional data needed by the Visualization Service to display the results to the user. In the example above the `dc:title` property is used to provide titles for the learning resources to facilitate displaying proper titles to the user and not just the links to the resources.

Defining an interface to the Personalization Service

The interface to the Personalization Service is more difficult to define than the interface to the Visualization Service because the Visualization Service in most cases is designed and implemented with respect to the whole Personal Reader architecture while the Personalization Service often is designed as a stand-alone service without any knowledge of the Personal Reader at all. Normally every Personalization Service has its own special input document specification and it is the responsibility of the Connector Service to create this specific document for each Personalization Service connected.

Therefore the interface is defined in an abstract way through the steps the Connector Service has to perform for the Personalization Service as they are listed at the beginning of this chapter. An example interface implementation is discussed in the implementation chapter.

2.3 The Visualization/User Service

The Visualization Service provides the interface to the user or learner. Every interaction with the user takes places via this service. The User Service is responsible for handling user authentication and maintaining user and course databases. In this paper however we assume that the Visualization Service implementation provides both functionalities. If one functionality is not provided by the implementation an additional service is needed but this is not discussed here.

In our scenario the Visualization Service is responsible for:

- displaying a login
- handling user authentication
- getting input from the user
- sending requests via the Connector Service
- displaying the results in a suitable form to the user
- updating the user and course database

The implementation independent parts of the interface to the Connector Service are defined in the preceding chapter. The missing parts, like the Java interface, are discussed together with the example implementation in the implementation chapter.

The Visualization Service may employ whatever techniques are most suitable for its domain, e.g. a web-based service or normal application will surely use different techniques and standards for displaying the results and handling the user input. Other examples for different usage domains, which require different Visualization Service implementations, could be:

- PDAs or cell phones
- special implementations for blind people
- implementations for different user groups, e.g. young kids, adults etc.

One important task for the Visualization Service is proper localization. Especially in a learning environment the framework provided by the service for navigation etc., not necessarily the content itself, should be available in the learner's native language.

The implementation chapter discusses an example Visualization Service implementation based on Java Servlets, Java Server Pages (JSP) and other web standards.

2.4 The Personalization Service

Each personalization service possess reasoning rules for some specific adaptation purposes. These rules query for resources and metadata, and reason over distributed data and metadata descriptions. A major step for reasoning after having queried the user profile, the domain ontology, and learning objects is to construct a temporally valid task knowledge base as a base for applying the adaptation rules.

For implementing the reasoning rules, we decided to use the TRIPLE query and rule language for the Semantic Web [Sintek and Decker, 2002]. Rules defined in TRIPLE can reason about RDF-annotated information resources (required translation tools from RDF to triple and vice versa are provided). An RDF statement (which is a triple) is written as `subject[predicate -> object]`

RDF *models* are explicitly available in TRIPLE: Statements that are true in a specific model are written as ”@model”. This is particularly important for constructing the *temporal knowledge bases* as required in the Personal Reader. Connectives and quantifiers for building logical formulae from statements are allowed as usual: `AND`, `OR`, `NOT`, `FORALL`, `EXISTS`, `<-`, `->`, etc. are used.

3 Implementation

This chapter discusses our example implementations of the Personal Reader framework:

- a JAX/XML-RPC based Connector Service
- a Java Servlet/JSP based Visualization Service

Finally we present some examples for personalization functionality using TRIPLE as Personalization Service.

3.1 The Connector Service

The Connector Service implementation presented in this paper uses JAX-RPC and XML-RPC, which are both XML-based RPC techniques, to communicate with the Visualization and Personalization Service instances.

The RDF document returned by the Connector Service is discussed in the architecture chapter, in the following we will describe the more implementation dependent parts of these interfaces.

Implementing the interface to the Visualization Service

The Connector Service runs in a web-service container, e.g. Apache Tomcat, and is implemented in the Java programming language. The interface to the Visualization Service is defined through the following Java interface:

```
public interface CServiceIF extends Remote {

public String getAnnotatedDescription(
String pageURL,
String userProfileAsRDF,
String descriptionURL,
String ontologyURL)
throws RemoteException;

}
```

`pageURL` This parameter contains the URL of the requested learning resource.

`userProfileAsRDF` This parameter contains the user profile as presented in the architecture chapter in a RDF/XML representation serialized to a string.

`descriptionURL` This parameter contains the URL of the RDF document with the learning resource description as presented in the architecture chapter.

`ontologyURL` This parameter contains the URL of the RDF document with the domain ontology as presented in the architecture chapter.

To access the Connector Service via this interface using JAX-RPC, a WSDL description of the service is needed, which specifies the Java interface to use as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration xmlns="http://.../xml/ns/jax-rpc/ri/config">
  <service name="CService" packageName="de.l3s.cs"
      targetNamespace="http://.../wsdl"
      typeNamespace="http://.../types">
    <interface name="de.l3s.cs.CServiceIF"/>
  </service>
</configuration>
```

The specified interface then needs to be mapped to a service endpoint, which provides an URL mapping and a class implementing the interface:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<webServices xmlns="http://.../ns/jax-rpc/ri/dd" version="1.0"
  targetNamespace="http://.../wsdl"
  typeNamespace="http://.../types">

  <endpoint name="CService" interface="de.l3s.cs.CServiceIF"
    model=".../model.gz" implementation="de.l3s.cs.CService"/>

  <endpointMapping endpointName="CService" urlPattern="/cs"/>

</webServices>
```

The service now can be called from JAX-RPC enabled applications just like a normal Java method.

Impl. the interface to the Personalization Service

As shown in the preceding section the class `CService` implements the service interface `CServiceIF` to the Visualization Service. The business-logic for performing the steps needed to create the Personalization Service specific interface documents, in our example implementation for the TRIPLE based Personalization Service, is encapsulated into so-called *Connector* classes.

The *Connector* class for the TRIPLE service is called `TripleConnector` and uses the Jena 2 framework, a framework for writing Semantic Web applications focused on RDF, to process the input documents from the Visualization Service and the documents fetched from the web, like the learning resource description and the domain ontology, and the output document received from the Personalization Service and to create the result document for the Visualization Service.

Here is a code fragment from the `TripleConnector` showing how the TRIPLE input document is constructed:

```
query = ModelFactory.createDefaultModel();

Resource myQuery = query.createResource(QUERY_URL);
Resource pageURLAsResource = query.createResource(pageURL);

myQuery.addProperty(RDF.type, L3S.Query);
myQuery.addProperty(L3S.aboutUser, user);
myQuery.addProperty(L3S.currentPage, pageURLAsResource);
myQuery.addProperty(L3S.queryFor, L3S.general);
myQuery.addProperty(L3S.queryFor, L3S.details);
myQuery.addProperty(L3S.queryFor, L3S.summary);
myQuery.addProperty(L3S.queryFor, L3S.quiz);
myQuery.addProperty(L3S.queryFor, L3S.recommended);
myQuery.addProperty(L3S.queryFor, L3S.notRecommended);
myQuery.addProperty(L3S.queryFor, L3S.alreadyVisited);

StringBuffer tripleProgBuff = new StringBuffer();

tripleProgBuff.append(reasonerAsTriple);
tripleProgBuff.append("\n");

tripleProgBuff.append("@'http://.../test#':user {\n");
tripleProgBuff.append(
  new TRIPLEDump(userProfile, "abbrev.txt").dumpToString());
tripleProgBuff.append("\n");
tripleProgBuff.append("}\n");
tripleProgBuff.append("\n");

tripleProgBuff.append("@'http://.../test#':query {\n");
tripleProgBuff.append(
  new TRIPLEDump(query, "abbrev.txt").dumpToString());
tripleProgBuff.append("\n");
tripleProgBuff.append("}\n");
tripleProgBuff.append("\n");

tripleProgBuff.append(
  "@'http://www.examples.org/test#':sun_tutorial {\n");
tripleProgBuff.append(descriptionAsTriple);
tripleProgBuff.append("\n");
tripleProgBuff.append("}\n");
tripleProgBuff.append("\n");

tripleProgBuff.append(
  "@'http://.../java_ontology.rdf':java_ontology {\n");
tripleProgBuff.append(ontologyAsTriple);
tripleProgBuff.append("\n");
tripleProgBuff.append("}\n");
tripleProgBuff.append("\n");

triple = ModelFactory.createDefaultModel();
triple.read(
  new StringReader(
    (String) new TRIPLEClient(TRIPLE_HOST, TRIPLE_PORT)
      .query(tripleProgBuff.toString(), "")
      .get("rdf")),
    null);
```

In the upper part of the code a RDF query is constructed, which specifies the personalization functionality wanted from the Personalization Service. Here is an example how this query may look like:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://.../l3s.rdf#" >
<rdf:Description rdf:about="http://.../l3s.rdf#myQuery">
 <rdf:type rdf:resource="http://.../l3s.rdf#Query"/>
 <j.0:aboutUser rdf:resource="http://.../user#john"/>
 <j.0:currentPage rdf:resource="http://.../variables.html"/>
 <j.0:queryFor rdf:resource="http://.../l3s.rdf#general"/>
 <j.0:queryFor rdf:resource="http://.../l3s.rdf#details"/>
 <j.0:queryFor rdf:resource="http://.../l3s.rdf#summary"/>
 <j.0:queryFor rdf:resource="http://.../l3s.rdf#quiz"/>
 <j.0:queryFor rdf:resource="http://.../l3s.rdf#recommended"/>
 <j.0:queryFor rdf:resource="http://.../l3s.rdf#notRecommended"/>
 <j.0:queryFor rdf:resource="http://.../l3s.rdf#alreadyVisited"/>
</rdf:Description>
</rdf:RDF>
```

Every query is of the type `Query` and contains the following properties:

- `aboutUser`
- `currentPage`
- `queryFor`

`aboutUser` This property specifies the user/learner the query requests personalization functionality for.

`currentPage` This property specifies the learning resource the query is about.

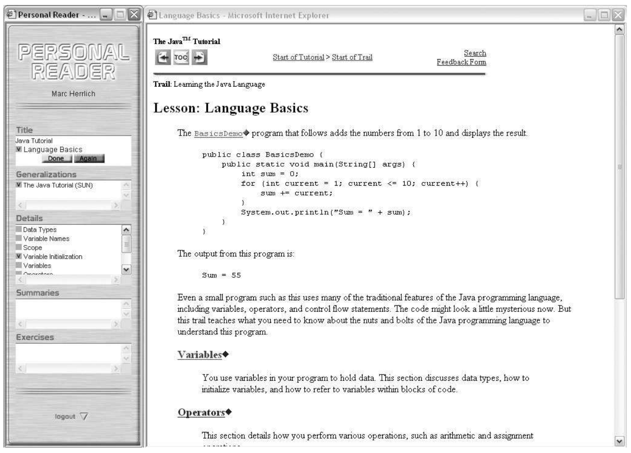
`queryFor` This property specifies the personalization functionalities requested. For every functionality needed there must be one `queryFor` property referring to a well-defined RDF resource. The resources used in the example query above are the same as shown in the preceding chapter with the result document example and have the same meaning.

The lower part of the code fragment above shows how the query and the rest of the input documents are converted and aggregated into a single TRIPLE file to be send to the TRIPLE service. Then a new Jena model is created from the result received from the TRIPLE service and parsed into the result document already discussed (not in the code fragment above).

3.2 The Visualization Service

The Visualization Service implementation presented here is based on Java Servlets and Java Server Pages (JSP). It runs in a Java Servlet container like Apache Tomcat and uses JAX-RPC to access the Connector Service implementation presented in the preceding chapter. Java Beans are used to encapsulate the business-logic and to instantiate JSP templates, which generate HTML pages that can be displayed to the user using a standard web-browser like Mozilla or the Internet Explorer.

Figure 2 depicts a screenshot showing how learning resources are presented to the user.



The Java™ Tutorial

Trail: Learning the Java Language

Lesson: Language Basics

The BasicsDemo◆ program that follows adds the numbers from 1 to 10 and displays the result.

```
public class BasicsDemo {
    public static void main(String[] args) {
        int sum = 0;
        for (int current = 1; current <= 10; current++) {
            sum += current;
        }
        System.out.println("Sum = " + sum);
    }
}
```

The output from this program is:

```
Sum = 55
```

Even a small program such as this uses many of the traditional features of the Java programming language, including variables, operators, and control flow statements. The code might look a little mysterious now. But this trail teaches what you need to know about the nuts and bolts of the Java programming language to understand this program.

Variables◆

You use variables in your program to hold data. This section discusses data types, how to initialize variables, and how to refer to variables within blocks of code.

Operators◆

This section details how you perform various operations, such as arithmetic and assignment

Figure 2: Screenshot of a Personal Reader for learning resources on Java programming

In the right part of the screen the learning resource itself is displayed. The left part displays the visualization of the results received via the Connector Service.

3.3 The Personalization Service

In the following, we will describe two examples of personalization functionality in the Personalization Service.

Generating Links

Generating links to more detailed learning resources is an adaptive functionality in this example Personal Reader.

The adaptation rule takes the isA hierarchy in the domain ontology, in this case the domain ontology for Java programming, into account to determine domain concepts which are details of the current concept or concepts that the learner is studying on the learning resource. In particular, more details for the currently used learning resource is determined by `detail_learningobject(LO, LO_DETAIL)` where `LO` and `LO_Detail` are learning resources, and where `LO_DETAIL` covers more specialized learning concepts which are determined with help of the domain ontology.

```
FORALL LO, LO_DETAIL detail_learningobject(LO, LO_DETAIL) <-
  EXISTS C, C_DETAIL(detail_concepts(C, C_DETAIL)
    AND concepts_of_LO(LO, C)
    AND concepts_of_LO(LO_DETAIL, C_DETAIL))
    AND learning_resource(LO_DETAIL) AND NOT unify(LO,LO_DETAIL).
```

N. B. the rule does neither require that `LO_DETAIL` covers all specialized learning concepts, nor that it exclusively covers specialized learning concepts. Further refinements of this adaptation rule are of course possible.

Calculating Recommendations

Recommendations are personalized according to the current learning progress of the user, e. g. with respect to the current set of course materials. The following rule determines that a learning resource `LO` is `recommended` if the learner studied at least one more general learning resource (`UpperLevelLO`):

```
FORALL LO1, LO2 upperlevel(LO1,LO2) <-
  LO1['http://purl.org/dc/terms#':isPartOf -> LO2].

FORALL LO, U learning_state(LO, U, recommended) <-
  EXISTS UpperLevelLO (upperlevel(LO, UpperLevelLO) AND
                      p_obs(UpperLevelLO, U, Learned) ).
```

Additional rules deriving stronger recommendations (e. g., if the user has studied *all* general learning resources), less strong recommendations (e.g., if one or two of these haven't been studied so far), etc., are possible, too.

Recommendations can also be calculated with respect to the current domain ontology. This is necessary if a user is regarding course materials from different courses at the same time.

```
FORALL C, C_DETAIL detail_concepts(C, C_DETAIL) <-
 C_DETAIL['http://.../2000/01/rdf-schema#':subClassOf -> C]
 AND concept(C) AND concept(C_DETAIL).

FORALL LO, U learning_state(LO, U, recommended) <-
 EXISTS C, C_DETAIL (concepts_of_LO(LO, C_DETAIL)
  AND detail_concepts(C, C_DETAIL) AND p_obs(C, U, Learned) ).
```

However, the first recommendation rule, which reasons within one course will be more accurate because it has more fine–grained information about the course and therefore on the learning process of a learner taking part in this course. Thus, our strategy is to apply first the adaptation rule which take most observations and data into account, and, if these rules cannot provide results, apply less strong rules. In future work, we will extend this approach. Currently, we are considering in enriching the results of the rules with confidence parameters. How these confidence values can be smoothly integrated into a user interface is an open research question.

4 Related work

Related work to our approach can be found in recent personalization systems like e.g. [Frasincar and Houben, 2002; Conlan et al., 2003], and personalized learning portals [Brusilovsky and Nijhawan, 2002; Brusilovsky, 2004].

[Frasincar and Houben, 2002] focuses on content adaptation, or, more precisely, on personalizing the presentation of hypermedia content to the user. Adaptability is provided by certain adaptability conditions, e. g., the ability of a device to display images. Adaptivity is based on the AHAM idea [Bra et al., 1999] of event–conditions for resources: A presentation / fragment is desirable if its appearance condition evaluates to true. [Dagger et al., 2003] builds on separating learning resources from sequencing logic and additional models for adaptivity: Adaptivity blocks are defined in the metadata of learning objects. Candidate groups and components define which kind of adaptivity can be realized on the current learning content. Driving force in these models are the candidate groups that define how to teach a certain learning concept. A rule engine selects the best candidates for each user in a given context. Adaptivity requirements are considered only in the adaptivity blocks.

Personalized learning portals are investigated in [Brusilovsky and Nijhawan, 2002]. The learning portals provide views on learning activities which are provided by so–called *activity servers*. The activity servers store both learning content and the learning activities possible with this special content. A central student model server collects the data about student performance from each activity server the student is working on, as well as from every portal the student is registered to. In [Brusilovsky, 2004], also *value-added services* are introduced in the architecture. In our approach, we follow a rigorous approach to separate structural information and personalization information from the real web content [Henze and Nejdl, 2003]. Personalization functionality is then exposed as a service - which the user can use for getting a personalized interface: a personal reader for web content. Compared to the learning portal approach in [Brusilovsky, 2004] we implement value–added services.

5 Conclusion

In this paper, we have presented a framework for designing, implementing and maintaining adaptive *reader* applications for the Semantic Web. The Personal Reader framework is based on the idea of establishing personalization functionality as services on the Semantic Web. We have discussed the architectural outline of the Personal Reader, and have shown how services orchestration is realized. The implementation of a reader for the Sun Java Programming tutorial has been discussed as an example. Currently, we are using the framework to design a Reader for publications, and are investigating how learner assessment can be integrated to enhance Readers for learning resources. The current state of the project can be followed at `www.personal-reader.de`.

References

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*.

[Bra et al., 1999] Bra, P. D., Houben, G.-J., and Wu, H. (1999). AHAM: A dexter-based reference model for adaptive hypermedia. In *ACM Conference on Hypertext and Hypermedia*, pages 147–156, Darmstadt, Germany.

[Brusilovsky, 2004] Brusilovsky, P. (2004). KnowledgeTree: A distributed architecture for adaptive e-learning. In *Proceedings of the Thirteenth International World Wide Web Conference*, New York.

[Brusilovsky and Nijhawan, 2002] Brusilovsky, P. and Nijhawan, H. (2002). A framework for adaptive e-learning based on distributed re-usable learning activities. In *In Proceedings of World Conference on E-Learning, E-Learn 2002*, Montreal, Canada.

[Campione and Walrath, 2003] Campione, M. and Walrath, K. (2003). The java tutorial. http://java.sun.com/docs/books/tutorial/.

[Conlan et al., 2003] Conlan, O., Lewis, D., Higel, S., O'Sullivan, D., and Wade, V. (2003). Applying adaptive hypermedia techniques to semantic web service composition. In *International Workshop on Adaptive Hypermedia and Adaptive Web-based Systems (AH 2003)*, Budapest, Hungary.

[Dagger et al., 2003] Dagger, D., Wade, V., and Conlan, O. (2003). Towards "anytime, anywhere" learning: The role and realization of dynamic terminal personalization in adaptive elearning. In *Ed-Media 2003, World Conference on Educational Multimedia, Hypermedia and Telecommunications*, Hawaii.

[Dublin Core, 2004] Dublin Core (2004). Dublin Core. http://dublincore.org/.

[Frasincar and Houben, 2002] Frasincar, F. and Houben, G. (2002). Hypermedia presentation adaptation on the semantic web. In *Proccedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2002)*, Malaga, Spain.

[Henze and Nejdl, 2003] Henze, N. and Nejdl, W. (2003). Logically characterizing adaptive educational hypermedia systems. In *International Workshop on Adaptive Hypermedia and Adaptive Web-based Systems (AH 2003)*, Budapest, Hungary.

[Java Server Pages, 2004] Java Server Pages (2004). Java server pages. http://java.sun.com/products/jsp/.

[LOM, 2002] LOM (2002). LOM: Draft Standard for Learning Object Metadata. http://ltsc.ieee.org/wg12/index.html.

[RDF, 2002] RDF (2002). Resource Description Framework (RDF) Schema Specification 1.0. http://www.w3.org/TR/rdf-schema.

[Sintek and Decker, 2002] Sintek, M. and Decker, S. (2002). TRIPLE - an RDF Query, Inference, and Transformation Language. In Horrocks, I. and Hendler, J., editors, *International Semantic Web Conference (ISWC)*, pages 364–378, Sardinia, Italy. LNCS 2342.

[XML, 2003] XML (2003). XML: extensible Markup Language. http://www.w3.org/XML/.

[XML-based RPC, 2004] XML-based RPC (2004). XML-based RPC: Remote procedure calls based on xml. http://java.sun.com/xml/jaxrpc/index.jsp.