

Configurable Web Services and the Personal Reader Agent

Personal Reader Team

July 5th 2006

The Personal Reader Framework enables us to develop and maintain Web Content Reader. In this architecture Web Services are primarily used as providers of RDF data, the content which is presented the end user of a Web Content Reader. With our new approach of Configurable Web Services we allow users to configure the data providing Web Services. Such configurations can be stored and reused at a later date. Thereby the Personal Reader Agent is the interface between Users and Configurable Web Services. The Agent allows selection, configuration and calling of the Web Services and further provides personalization functionalities like reuse of stored configurations which suit to the users interests.

Contents

1	Introduction	3
2	The Configuration Ontology	3
3	Configurable Web Services	6
3.1	MyEar Music Web Service	6
3.1.1	Configurable description of the MyEar Music Web Service	6
3.1.2	Internals of the MyEar Music Web Service	7
3.1.3	MyEar View	9
3.2	Personal Publication Reader Web Service	9
3.2.1	Core Functionality of the Personal Publication Reader Web Service	9
3.2.2	Configurable description of PPR Web Service	9
3.2.3	Special View for Results of PPR Web Service	10
4	Personal Reader Agent	10
4.1	Core Functionality	11
4.2	Personalization Functionality	14

1 Introduction

Within the Personal Reader project we already developed Web Content Reader like the *Personal Publication Reader*[2] which allows browsing publications in an embedded context. We also utilized and extended the SWAD-E Semantic Portal software[9] to provide a Personal Semantic Portal[5]. Whereas these approaches are fixed in terms of the type of data that is provided, we now introduce a more generic approach: Configurable Web Services and the Personal Reader Agent. The Personal Reader Agent is a Web Application which enables users to select, configure and call Configurable Web Services. These Semantic Web Services need a detailed description of how they can be configured and how they are accessible. According to this description the Personal Reader Agent generates an interface that allows to adjust the web services. The Agent further provides some personalization functionalities like reuse of stored configurations of web services which suit to the users interests. This article is structured as follows: In section 2 we introduce the *Configuration Ontology*, the ontological grounding of our approach. After introducing this ontology we can figure out Configurable Web Services considering a concrete Web Service as example (section 3). In section 4 we outline the architecture and implementation of the Personal Reader Agent. Finally we conclude our work and point out reasonable next steps (section 5).

2 The Configuration Ontology

The Configuration Ontology defines on the one hand the vocabulary that is needed to describe a Configurable Web Service and on the other hand the concepts that are required for the personalization functionalities.

Explanations for figure 1:

1. Core Configurable Vocabulary (needed to describe a Configurable Web Service):

Configurable Each Configurable Web Service defines one instance of this class. Therefor a **name** and a **description** has to be defined. Example:

```
(#MyEarConfigurable, name, "MyEar Configurable")
(#MyEarConfigurable, description, "Configurable things of my MyEar Music Web Service")
```

ConfigurableItem A Configurable consists of several ConfigurableItems. Example:

```
(#MyEarConfigurable, hasConfigurableItem, #DurationItem)
(#DurationItem, name, "Duration")
(#DurationItem, description, "Duration of a Song that should be taken into account by my Web Service.")
```

Input Every ConfigurableItem has at least one Input. We define two special Inputs: a **SelectionInput**, which allows only predefined values, and a **TextInput**, which allows arbitrary values. For an Input a **type**, a **minNumber** and a **maxNumberOfInputValues** has to be specified. Example:

```
(#DurationItem, input, #MinDurationInput)
(#MinDurationInput, description, "The minimum duration of a song (in minutes)")
(#MinDurationInput, type, http://www.w3.org/2001/XMLSchema#nonNegativeInteger)
```

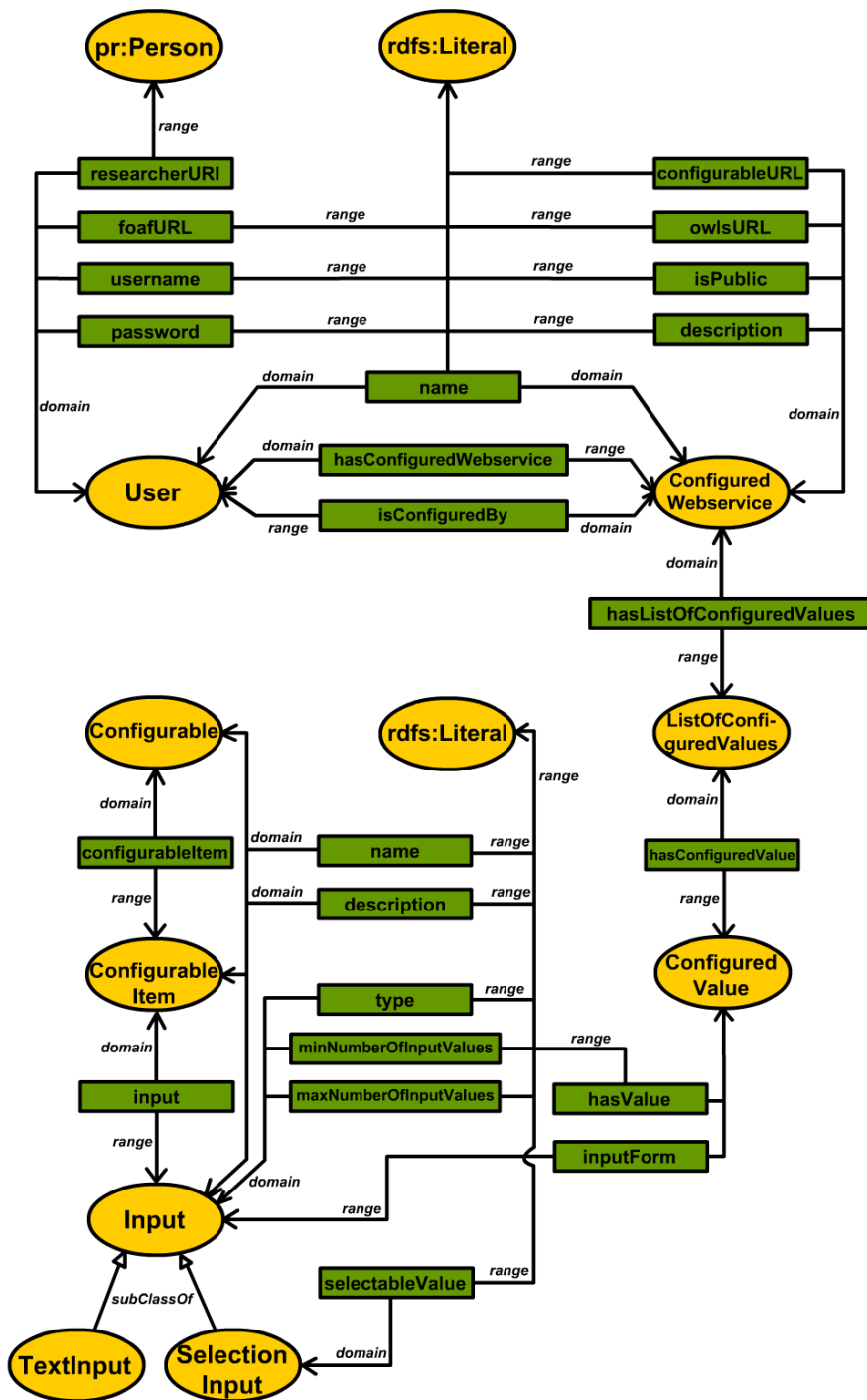


Figure 1: Configuration Ontology

```
(#MinDurationInput, minNumberOfInputValues, 0)
(#MinDurationInput, maxNumberOfInputValues, 1)

(#DurationItem, input, #MaxDurationInput)
...
```

2. User and their configured Web Services (needed to realize personalization functionalities):

User This concept models the users of our *Personal Reader Agent*. Therefore a **User** is featured with an **username**, **password**, **name** and a list of **ConfiguredWebservices** (**hasConfiguredWebservice**). To guarantee interconnection to other concepts that model a user we define the properties **researcherURI**, which points to the corresponding instance within our *Researcher Ontology*¹, and **foafURL**, which links the FOAF² description of the user. Example:

```
(#abelFabian, username, "fob")
(#abelFabian, password, "secret")
(#abelFabian, name, "Fabian Abel")
(#abelFabian, foafURL, "http://www.fabianabel.de/fabian.abel.rdf")
(#abelFabian, researcherURI, http://www.personal-reader.de/reverse#abelFabian)
(#abelFabian, hasConfiguredWebservice, #abelFabianMyEarJazzConfigWS)
(#abelFabian, hasConfiguredWebservice, #abelFabianPPRBioInformaticsConfigWS)
...
```

ConfiguredWebservice This concept is used to store configurations of Web Services made by a user. The properties **name** and **description** allow to describe the concrete configuration. The boolean property **isPublic** indicates whether a **ConfiguredWebservice** can be accessed and re-used by other users than the user who configured it (**isConfiguredBy**). **owlsURL** points to the OWL-S[6] description of the Web Service that was configured by the user and **configurableURL** points to the **Configurable** description. The values that belong to the concrete configuration are listed within the **ListOfConfiguredValues**. Example:

```
(#abelFabianMyEarJazzConfigWS, name, "Jazz Music")
(#abelFabianMyEarJazzConfigWS, description, "This configuration of the MyEar Music Web
Service effects the Web Service to aggregate
podcasting items that are related with Jazz.")

(#abelFabianMyEarJazzConfigWS, isPublic, "true")
(#abelFabianMyEarJazzConfigWS, isConfiguredBy, #abelFabian)
(#abelFabianMyEarJazzConfigWS, owlsURL, "...MyEar/rdf/MyEarOWLS.owl")
(#abelFabianMyEarJazzConfigWS, configurableURL, #MyEarConfigurable)
(#abelFabianMyEarJazzConfigWS, hasListOfConfiguredValues, #abelFabianMyEarJazzValueList)
```

ListOfConfiguredValues This is a list of the values that are configured by a user. Each **ConfiguredValue** has a **value** (range: typed Literals) and a reference to the **Input** (**inputForm**) which defines what is applicable in general. Example:

```
(#abelFabianMyEarJazzValueList, hasConfiguredValue, #abelFabianMyEarJazzValue1)
(#abelFabianMyEarJazzValue1, value, "3")
(#abelFabianMyEarJazzValue1, inputForm, #MinDurationInput)
(#abelFabianMyEarJazzValueList, hasConfiguredValue, #abelFabianMyEarJazzValue2)
...
```

¹The Researcher Ontology models the organizational structure of the REWERSE project: <http://www.personal-reader.de/rdf/ResearcherOntology.owl>

²Friend of a Friend: <http://www.foaf-project.org/>

3 Configurable Web Services

In the context of the *Personal Reader Agent* we use Configurable Web Services as RDF data provider. Such Web Services has to satisfy three requirements:

1. they need a Configurable description according to section 2
2. they have to be Semantic Web Services which means that they have a OWL-S description
3. they must be able to process RDF data (more precisely: `ListOfConfiguredValues`, section 2) in order to understand a Web Service call (see section 2 for details)

Below we illustrate concepts and implementation issues of Configurable Web Services by presenting the *MyEar Music Web Service* (section 3.1) and the *Personal Publication Reader Web Service* (section 3.2).

3.1 MyEar Music Web Service

The MyEar Music Web Service enables users to listen to their personalized *podcasting feed*. A podcasting feed is in fact a RSS 2.0 feed[10] whose items refer to audio files. Our MyEar Music Web Service searches the web for podcasting feeds that suits to the users interest and then combines items from different feeds to present a personalized podcasting feed to the user.

3.1.1 Configurable description of the MyEar Music Web Service

A part of the Configurable description is outlined in the examples of section 2. Overall there are four *ConfigurableItems*:

myEarKeywordItem A keyword that should be within an item of a podcasting feed. `myEarKeywordItem` is defined as followed:

```
(#myEarKeywordItem, input, #myEarKeyword)
(#myEarKeyword, rdf:type, #TextInput)
(#myEarKeyword, minOccurs, 1)
(#myEarKeyword, description, "Enter at least one keyword that should be within an item of
                               a podcasting feed, e.g. Jazz, Classic,..")
(#myEarKeyword, type, http://www.w3.org/2001/XMLSchema#string)
...
```

myEarItunesCategoryItem This item refers to a `SelectionInput` which only permits the selection of values that correspond to *itunes:category*[1]:

```
(#myEarItunesCategoryItem, input, #myEarCategory)
(#myEarCategory, rdf:type, #SelectionInput)
(#myEarCategory, selectableValue, "Music")
(#myEarCategory, selectableValue, "Public Radio")
(#myEarCategory, selectableValue, "Arts")
...
(#myEarCategory, minOccurs, 0)
(#myEarCategory, type, http://www.w3.org/2001/XMLSchema#string)
...
```

myEarDurationItem The duration item has two inputs which allow to specify minimum and maximum duration of audio files that should be included into the personalized podcasting feed (see also section 2):

```
(#myEarDurationItem, input, #myEarMinDuration)
(#myEarDurationItem, input, #myEarMaxDuration)
...
```

maxNumberOfGoogleCallsItem For the search of applicable podcasting feeds we use the *Google SOAP Search API*[4]. According to the Google API terms we only get 10 results per requests and thus the MyEar Music Web Service has to call the Google Search Web Service several times. By enabling the user to configure the maximum number of Google calls the user can affect the runtime of the MyEar process.

3.1.2 Internals of the MyEar Music Web Service


If the MyEar Music Web Service is called by the Personal Reader Agent then there is used only one parameter: RDF data which embodies the `ListOfConfiguredValues` (see last example of section 2). At first this RDF has to be parsed in order to extract the configured values. After that the following method is called:


```
public String getMusic(String keyword, String musicCategory,
                      Date dateFrom, Date dateTill,
                      int minDuration, int maxDuration,
                      int maxNumOfGoogleCalls){
//call the Gogle API maxNumOfGoogleCalls-times with keyword as query
...
//for each result of the Google API call:
processGoogleResult(result, resultChannel, musicCategory,
                    dateFrom, dateTill,minDuration, maxDuration);
...
}
```

Within the method `processGoogleResult()` the result list of the Google API call is filtered: Only each `rss:item` that fulfils the configured restrictions (e.g. `musicCategory`,...) is added to the `resultChannel` (`rss:channel`).

When the MyEar Music Web Service has finished the search a `resultChannel` is returned:

```
<rss version="2.0" ...>
  <channel>
    <title>Results of search...</title>
    <item>
      <title>Jazzatronic 2005</title>
      <link>http://www.bendingcorners.com/2006/jazzatronic_2005/</link>
      <description>
        BendingCorner's annual set of the past year's finest "jazzatronic"
        tunes. A collection of jazz-based electronically-influenced groovers
        with modern production techniques.
      </description>
      <pubDate>Sat, 29 Apr 2006 10:27:00 CEST</pubDate>
      <guid isPermaLink="false">http://www.bendingcorners.com/rss.xml</guid>
      <enclosure url="http://www.bendingcorners.com/2006/jazzatronic-2005-lo.mp3"
        length="33662849" type="audio/mpeg" />
    </item>
  </channel>
</rss>
```





Personal Music Aggregation


My Ear

[\[about\]](#) [\[browse\]](#)

Browse the results of the MyEar Webservice

Starfrosch - Jazz
Channel: <http://www.starfrosch.ch/pod/jazz.xml>

Pamela's Parade - Les ventilateuses [insub02]



Pamela's Parade wurde 2000 in Genf als Drum und Sax Duo gegründet, später kam eine Posaune und ein Doublebass dazu.

Pamela's Parade ist inzwischen ein Jazz Quartet beeinflusst von Folk und Free Jazz oder pulsierendem Funk.

<http://www.dincise.net/insubordinations/>

Download

Date: Wed Feb 22 12:30:15 CET 2006
URL: http://www.archive.org/download/INSUB02/insub02_Pamelas_Parade_01_les_ventilateu...
Duration: 35339074

In The Groove, Jazz and Beyond
Channel: http://www.lasternet.com/inthegroove/ITG_podcast.xml
A weekly Jazz radio show that airs on WHUS 91.7FM, Storrs CT. From the Jazz masters of past and present to emerging new artists performing jazz, fusion and funk. No smooth jazz here!


Great 88's
Here is some great piano jazz from both well known masters as well as little known talented jazz pianists.

Date: Sun Apr 23 05:37:07 CEST 2006
URL: http://pod-serve.com/audiofile/filename/1120/itg_20060422.mp3
Duration: 41300000

Swing is in the Air for April 16, 2006

More new releases and reissues


Direct download: mp3 (62mb | 90min)



Date: Sun Apr 16 23:30:00 CEST 2006
URL: http://media.libsyn.com/media/radiojazz/swing_20060416.mp3
Duration: 64826230

ReadingCorner: jazz-n-groove

My Ear Player



Actual Playing:

name/1120/itg_20060422.mp3




Figure 2: MyEar View

```

    <title>A Tour of California Artists</title>
    ...
  </item>
  ...
</channel>
</rss>

```

Here each `rss:item` is extracted from a separate podcasting feed, e.g. the first item originates from a podcast which is located at: <http://www.bendingcorners.com/rss.xml>. Thus at the moment the `guid` element is estranged to link the podcast.

3.1.3 MyEar View

Results of the MyEar Web Service can be viewed in a standard RDF browser or in the *MyEar View* which is tailored for these results. A screenshot of this application is shown in figure 2. The *MyEar Player* on the right hand side directly allows users to listen to the audio files that are referenced by the items of the result channel.

3.2 Personal Publication Reader Web Service

3.2.1 Core Functionality of the Personal Publication Reader Web Service

The Personal Publication Reader (PPR) demonstrates how to provide personalized, syndicated views on distributed Web Data using Semantic Web technologies. The application comprises several steps: In first step, the information about different publications is extracted from distributed heterogeneous sources and enriched with machine-readable semantics. Next step would be reasoning step. In this step, rules reason about the created semantic descriptions and additional knowledge-bases like ontologies and user profile information. In last step which is actually user interface creation, the result of reasoning step in the shape of RDF is interpreted and translated into an appropriate, personalized user interface. In other words, with several simple mouse clicks, end user is able to browse all relevant information regarding an specific publication. Relevant information contains author's contact information, other publications of authors, different working groups, current publications in a specific working group and so on. For an online demo of PPR Web service, refer to [8].

3.2.2 Configurable description of PPR Web Service

As we have seen in previous sections, we need a Configuration-Description for a configurable Web service. Therefor we present a Configuration-Description for PPR. This Configuration-Description has been developed in two versions. The first version of Configuration-Description is based on the current functionalities of PPR. Generally, in current implementation of PPR, we send a RDF description including title of publication and PPR responses with all relevant information regarding this publication. The first version of PPR Configuration-Description has following main triples:

```
(#PPRConfigurable, hasConfigurableItem, #PublicationTitleItem)
```

```
(#PublicationTitleItem, input, #PublicationTitle)
(#PublicationTitle, description, "The title of the current publication")
(#PublicationTitle, type, http://www.w3.org/2001/XMLSchema#string)
(#PublicationTitle, minOccurs, 1)
(#PublicationTitle, maxOccurs, 1)
```

In the second version of Configuration-Description, we decided to use **Author name**, **Working group**, **Publication year**, and **Publication origin** as configurable items. In other words, we have following triples as configurable items:

```
(#PPRConfigurable2, hasConfigurableItem, #PublicationAuthorNameItem)
(#PPRConfigurable2, hasConfigurableItem, #PublicationWGItem)
(#PPRConfigurable2, hasConfigurableItem, #PublicationYearItem)
(#PPRConfigurable2, hasConfigurableItem, #PublicationOriginItem)
```

Each configurable item has its own properties. In a human language we can say: **#PublicationAuthorNameItem** has an input **AuthorName** and should occur exactly once in a request. **#PublicationWGItem** is a list of eight active working groups in REW-ERSE project. A request can contain at least zero and at most eight working groups. **#PublicationYearItem** has an input **Year** which indicates the publishing year. Finally, **#PublicationOriginItem** is a list of current twenty cities which publication information is gathered from their academic institutes.

3.2.3 Special View for Results of PPR Web Service

In an architectural overview, PPR is composed of three kinds of Web services: **Personalization services**, **Connector services**, and **Visualization services**. Visualization services are responsible for presenting the result of query to end use according to output device type and user settings. Beside RDF which is actually the raw format of output, the other supported format is currently HTML and it can be shown using any kind of Web browser. For an online demo of PPR application, refer to [8].

4 Personal Reader Agent

The Personal Reader Agent is on the one hand a kind of wizard that allows to select, configure and call the Configurable Web Services (*Core Functionality*) and on the other hand it provides the management of users and their saved configurations (*Personalization Functionality*).

For technical purpose the Agent is a J2EE Web Application that adheres to the *Model View Controller* approach. Figure 3 gives an overview of the Agent's architecture (or rather the package structure). General description of the Agent's packages:

reverse.agent The **AgentManger** is implemented as a singleton and is a specific extension of the session management. It memorizes objects that are relevant for the actual sessions like *selected Web Services of a user identified via SessionID*.

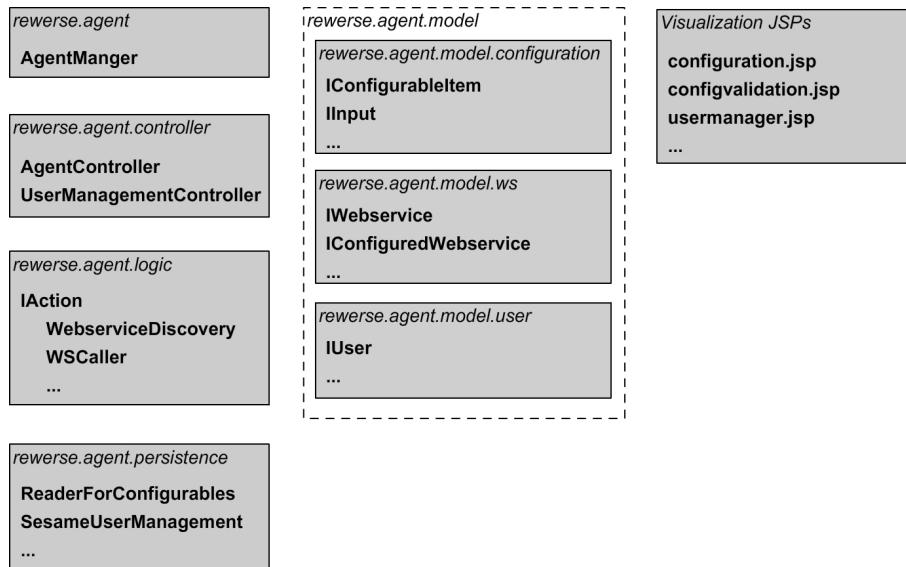


Figure 3: Architecture - Overview of Agent packages

rewerse.agent.controller This package contains the main controller (*HttpServlets*) of the Agent application: **AgentController** (responsible for core functionality) and **UserManagementController** (responsible for personalization functionality).

rewerse.agent.logic All actions like **WSCaller**, **WSConfigurationValidator** etc. implement the Interface **IAction**.

rewerse.agent.persistence The persistence package contains classes that..

- ... read in the OWL-S description and Configurable description of the Configurable Web Services and]
- ... provide access to a Sesame Repository[3] which stores user details and configurations made by users.

rewerse.agent.model Methods of the persistence layer return generally instances of the model package. We deal with three different kind of objects: Users, Web Services and Configurable descriptions. Thus we provide three packages containing Java models for these different types of objects.

Visualization JSPs Our presentation layer consists of a set of *Java Server Pages* (JSPs) which visualize the model objects that are the actual point of interest.

4.1 Core Functionality

The Controller that is responsible for the core functionality is the **AgentController**. It affords the following steps:

1. **Discovery** In this step the `AgentController` calls the `WebserviceDiscovery` action which requests the OWL-S descriptions of the Configurable Web Services that are registered at our simple UDDI. Finally the OWL-S descriptions are prepared for a selection by the users, according to figure 4.
2. **Configuration** After the first step the Agent reads in the Configurable descriptions of the selected Web Services (`ReaderForConfigurables`). Out of a corresponding Java object (`reverse.agent.model.configuration.IConfigurable`) a JSP generates HTML Forms that are illustrated in figure 5.
3. **Web Service Call** After all selected Web Services are configured without violating the restrictions defined in the corresponding Configurable descriptions (e.g. `maxNumberOfInputs`, `type`, ...), the Agent (`WSCaller`) is ready to call the Web Services (see figure 6). To ease the Web Service call we use the *OWL-S API*[7] provided by *mindswap*.
4. **Presenting the results** This step is not part of the Agent application but rather a task that can be done by a common RDF browser like *Piggy Bank*³ or an application that provides a special view for a certain RDF data (e.g. *MyEar View visualizes RSS 2.0 feeds*).

Step 1: Selecting a Webservice

Please select at least one of the following Webservices:

PPR
The PPR Webservice generates a personalized Research Information.

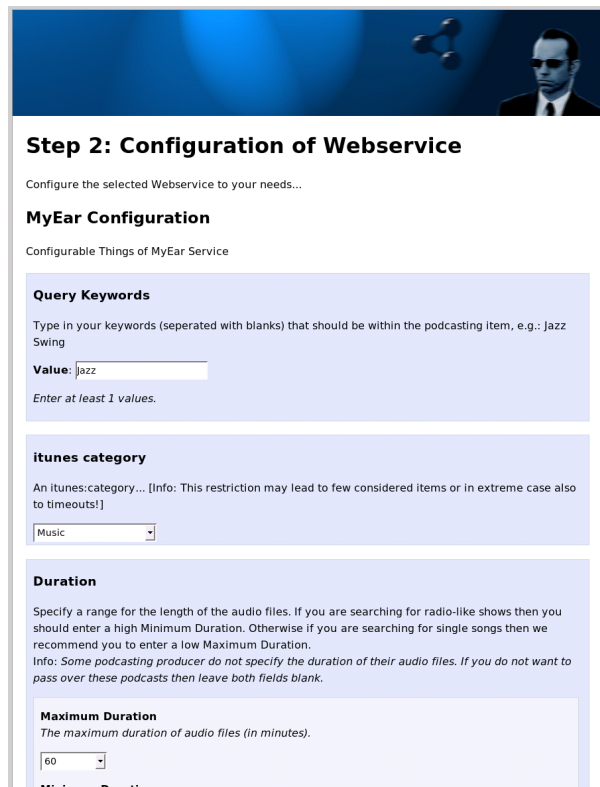
MyEar
The MyEar Webservice generates a personalized Podcasting feed.

Progress

1. Selection of Webservices 2. Configuration of Webservices 3. Browse Results

Figure 4: Step 1 - Selection of Configurable Web Services

³Piggy Bank - Firefox extension that allows browsing RDF data - <http://simile.mit.edu/piggy-bank/>



Step 2: Configuration of Webservice

Configure the selected Webservice to your needs...

MyEar Configuration

Configurable Things of MyEar Service

Query Keywords

Type in your keywords (seperated with blanks) that should be within the podcasting item, e.g.: Jazz Swing

Value:

Enter at least 1 values.

itunes category

An itunes:category... [Info: This restriction may lead to few considered items or in extreme case also to timeouts!]

Duration

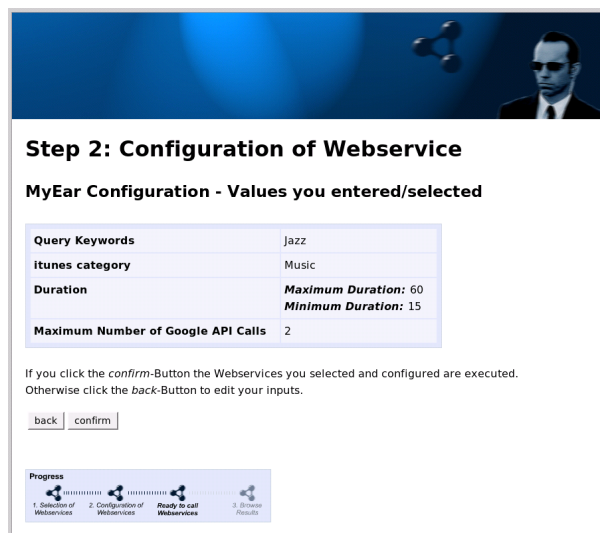
Specify a range for the length of the audio files. If you are searching for radio-like shows then you should enter a high Minimum Duration. Otherwise if you are searching for single songs then we recommend you to enter a low Maximum Duration.

Info: Some podcasting producer do not specify the duration of their audio files. If you do not want to pass over these podcasts then leave both fields blank.

Maximum Duration
The maximum duration of audio files (in minutes).

Minimum Duration

Figure 5: Step 2 - Configuration of Web Services



Step 2: Configuration of Webservice

MyEar Configuration - Values you entered/selected

Query Keywords	Jazz
itunes category	Music
Duration	Maximum Duration: 60 Minimum Duration: 15
Maximum Number of Google API Calls	2

If you click the *confirm*-Button the Webservices you selected and configured are executed. Otherwise click the *back*-Button to edit your inputs.

Progress

1. Selection of Webservices 2. Configuration of Webservices **Ready to call Webservices** 3. Behind Results

Figure 6: Step 3 - Ready to call Web Services

• Click on the "confirm + save"-Button to confirm and save your entries. Saving allows you to re-use your configuration at a later date.

MyEar Configuration - Values you entered/selected

Duration	Maximum Duration: 20 Minimum Duration: -
Maximum Number of Google API Calls	2
itunes category	Music
Query Keywords	Jazz

If you click the confirm-Button the Webservices you selected and configured are executed. Otherwise click the back-Button to edit your inputs.

Save your configuration of the Webservice

Saved configurations can be used at a later date to call the Webservice in a personalized way. It is also possible to adjust your configuration later. You have to enter at least a significant name for your configuration. Further you can enter a short description and select whether other users can utilize your configuration or not (standard is not).

Name: (e.g. MyEar - Jazz)

Description: (e.g. With this configuration the MyEar-Webservice returns a Jazz-Podcasting Feed...)

publish?: (If checked then other users can utilize your configuration)

Progress

1. Selection of Webservices 2. Configuration of Webservices 3. Ready to call Webservices 4. Entered Results

Figure 7: Saving Configurations - Entry of meta description about a configured Web Service

4.2 Personalization Functionality

Registered users have some options which lead to a more personalized Agent, they are enabled to...

...save configurations: As in section 2 outlined the ontological model behind *saved configurations* is *ConfiguredWebservice*. At this users can specify name, description and *isPublic* on their own, *owlsURL* and *configurableURL* is set by the Agent and the *ListOfConfiguredValues* arises from the configuration step which is also performed by the users. The HTML input form that allows entry of such meta information about configured Web Services is presented in figure 7.

...re-use their own configurations: In order to allow users a faster access to the Configurable Web Services they can call these services also with a saved configuration as illustrated in figure 8. Further the Agent provides some management functionality for configured Web Services (*view*, *edit* and *delete*).

...re-use recommended configurations of other users: If a *ConfiguredWebservice* is marked as *isPublic* then it can be re-used also by other users than the author. Therefor the Agent allows the listing of configurations that might be relevant for a user. To determine relevant configurations the Agent utilizes relations between users that are defined inside the *Researcher Ontology* or *FOAF* description. The Agent proceeds as follows:

Your Configured Web Services

List of Web Services you have configured using the Personal Reader Agent is shown below. You have the opportunity to view, edit or delete your Configured Web Services. Further you can re-use a Configured Web Service by clicking on call.

Name	Description	Is Public?	Actions
Die Testkonfiguration	Dies ist eine Testkonfiguration, die nichts bewirkt...	false	view edit delete call
MyEar - Jazz	A simple search for podcasting feeds related with Jazz...	true	view edit delete call
Rock - MyEar	A bissel Rock...	true	view edit delete call

If you have problems then mail us: readerteam@kbs.uni-hannover.de

Figure 8: List of configured Web Services

Relation within Researcher Ontology: The Researcher Ontology defines persons and their involvements in working groups. If two persons (users) are involved in the same working group then the Agent suggests that configurations made by *User A* are also interesting for *User B*.

Relation within FOAF description: FOAF defines among other things the relation (*Person, knows Person*). This relation can be used by the Agent to list configurations of other persons the user knows.

5 Conclusion and Outlook

With the Personal Reader Agent and Configurable Web Services we provide a dynamic approach to aggregate RDF data. With the Agent's personalization functionality we also allow to personalize the access to the Configurable Web Services.

Based on the current state of implementation there are still some open issues and conceivable extensions... [to be continued]

References

- [1] Apple Computer, Inc.
Podcasting and iTunes: Technical Specification, 2006
<http://www.apple.com/itunes/podcasts/techspecs.html>
- [2] R. Baumgartner, N. Henze and M. Herzog
The Personal Publication Reader: Illustrating Web Data Extraction, Personalization and Reasoning for the Semantic Web. European Semantic Web Conference ESWC 2005, Heraklion, Greece, 2005
<http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2005/eswcbhh.pdf>
- [3] J. Broekstra, A. Kampman and F. van Harmelen
Sesame: A Generic Architecture for Storing and Querying RDF, International Semantic Web Conference 2002, Sardinia, Italy, 2002
<http://openrdf.org/doc/papers/Sesame-ISWC2002.pdf>
- [4] Google
Google SOAP Search API, 2006
<http://www.google.com/apis/>
- [5] N. Henze and F. Abel
User Awareness and Personalization in Semantic Portals, International Semantic Web Conference, 2005
<http://reverse.net/publications/download/REWERSE-RP-2005-121.pdf>
- [6] D. Martin et. al.
OWL-S: Semantic Markup for Web Services, W3C Member Submission, 2004
<http://www.w3.org/Submission/OWL-S/>
- [7] mindswap, E. Sirin
OWL-S API, 2004
<http://www.mindswap.org/2004/owl-s/api/>
- [8] *Personal Reader Project*
<http://www.personal-reader.de/>
- [9] D. Reynolds, P. Shabajee, S. Cayzer and D. Steer
Semantic Portals Demonstrator - Lessons Learnt, SWAD-Europe deliverable 12.1.7, 2005
http://www.w3.org/2001/sw/Europe/reports/demo_2_report/
- [10] RSS Advisory Board
Really Simple Syndication 2.0 Specification, UserLand, 2002
<http://www.rssboard.org/rss-2-0/>