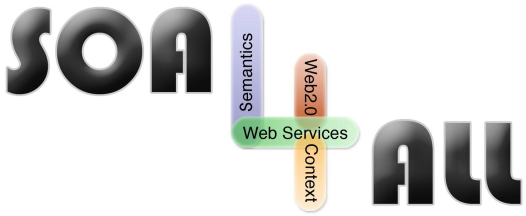

Project Number: **215219**

Project Acronym: **SOA4ALL**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D2.7.1 - Recommender System First Prototype

Activity:	Activity 1 - Fundamental & Integration activities	
Work Package:	WP2 - Service Deployment and Use	
Due Date:		M18
Submission Date:		11/09/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		CEFRIEL
Revision:		1.0
Authors:	Guillermo Álvaro Rey	iSOCO
	Dario Cerizza	CEFRIEL
	Giovanni Di Matteo	TXT
	Gianluca Ripa	CEFRIEL
	Andrea Turati	CEFRIEL
	Matteo Villa	TXT
Reviewers:	Sven Abels	TIE
	Nikolay Mehandjiev UNIMAN	

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	10/06/2009	Document Initialized	Dario Cerizza, Gianluca Ripa, Andrea Turati, Maurilio Zuccalà
0.2	22/06/2009	Inclusion of feedbacks on TOC	Dario Cerizza
0.3	30/06/2009	Contribution to Architecture	Dario Cerizza, Andrea Turati
0.4	06/07/2009	Contribution on chapter 2	Guillermo Álvaro Rey
0.5	15/07/2009	Document restructured according to prototype guidelines	Andrea Turati
0.6	17/07/2009	Inclusion of “Software Description” paragraph	Giovanni Di Matteo, Matteo Villa
0.7	20/07/2009	Described the installation and configuration of the RS	Andrea Turati
0.8	31/07/2009	Revisioned Software description and the whole paper in the annex. Described introduction and conclusion	Andrea Turati
0.9	06/08/2009	Described the test	Andrea Turati
0.10	07/08/2009	Described the Eclipse project	Andrea Turati
0.20	13/08/2009	Integrated comments from reviewer (Sven - TIE)	Dario Cerizza
0.21	25/08/2009	Integrated comments from reviewer (Nikolay – UniMan)	Dario Cerizza
1.0	31/08/2009	Last fixes	Dario Cerizza
1.0	11/09/2009	Final Editing	Malena Donato

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
Acronym	Full Name
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
BPEL	Business Process Execution Language
BPM	Business Process Modeling; Business Process Management
C2C	Consumer to Consumer
CMS	Content Management System
CP	Consumption Platform
D	Deliverable
DSB	Distributed Service Bus
EC	European Commission
eCommerce	Electronic Commerce
ESB	Enterprise Service Bus
EU	European Union
EXT GWT	Extended GWT
FOAF	Friend Of A Friend
FP	Framework Program
FP7	The 7th Framework Program
GUI	Graphical User Interface
GWT	Google Web Toolkit
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
PP	Provisioning Platform
RDF	Resource Description Framework
RS	Recommender System
SA-REST	Semantic Annotations for RESTful Services
SAWSDL	Semantic Annotations for WSDL
SOA	Service-Oriented Architecture
SOA4All	Service-Oriented Architectures for All

SOAP	Simple Object Access Protocol
SWS	Semantic Web Service
T	Task
WP	Work Package

Executive summary

This document complements the release of the first Recommender System (RS) prototype. This document accompanies the zip file that contains the first prototype of the Recommender System, as the result of the activities performed in the scope of T2.7. The zip file contains the source code, the installation and configuration facilities and the execution and testing facilities.

The first version of RS aims to support SOA4All users by providing suggestions (a.k.a. recommendations) about services that they may be interest in. The RS is based upon a collaborative filtering technique that exploit the behaviour of SOA4All users in interacting with the SOA4All studio and derive similarities among users. Thanks to this similarities, the RS is able to recommend to a user those service that have been used by similar users.

The body of this document follows the project guidelines for prototypes releases, thus it reports on the description of the component and the installation and configuration activities. Annex A provides an insight of the approach adopted, the architecture defined and the integration performed to provide a recommender system in SOA4All.

For the final prototype, we aim to further extended the RS by considering not only the similarities among users but also some similarities derivable from the semantic descriptions of the services.

1. Introduction

1.1 Purpose and Scope

This deliverable concerns the Recommender System (RS) integrated within the SOA4All Consumption Platform. This system aims to improve the user experience by providing users with suggestions about relevant services that may be of their interest. The RS analyzes user behaviour in interacting with the platform and exploits user similarities to provide recommendations.

M18 deadline is specifically focused in integrating a first version of the RS mainly based upon aggregating user behaviour. By M30, we aim to further extend the RS leveraging over the semantic descriptions of services and understanding how semantic relationships can improve the recommendations.

The goal of this deliverable is to complement the Recommender System prototype. This deliverable is included as part of the zip file, D271-P.zip, which contains the first software, source code, installation and configuration facilities, execution and testing facilities. We refer the reader to Annex A for details about the prototype architecture and design.

1.2 Structure of the document

First of all, in chapter 2 we introduce the RS by describing its role within the whole SOA4All architecture and its interaction with the other components. In addition, it shows how it is used by the scenarios defined in the use-cases. Then, chapter 3 describes how to install the RS and how to configure it for the first execution. Finally, chapter 4 provides some details about the software, describing how it is internally structured, illustrating the workflows that result in the invocation of the RS and presenting some expected screenshots showing the way in which the recommended items will be displayed. The document is followed by three annexes which present advanced aspects. Annex A includes a concise explanation of both the design and the internal architecture of the RS. In addition, it might be considered as a short paper representing a starting point for future dissemination activities. Annex B and Annex C report respectively the ontology schema used to model the actions of the users within the SOA4All Studio (representing the log produced by the SOA4All Studio) and an instance of possible user interactions (i.e. a possible log file).

2. General Overview of the Recommender System

The objective of this chapter is to illustrate the role of the Recommender System in SOA4All. In particular, we describe the relation between the Recommender System and the other main activities performed in the project, namely the overall architecture of SOA4All and the use cases. Please, refer to Annex A for details about the design of the internal architecture of the RS.

2.1 Relation with the SOA4All Architecture

Interacting with services in a service world as the one envisaged by SOA4All requires implementing several mechanisms in order to enhance the user experience within the vast number of services expected. In particular, enabling ways to help end-users interact with the most suitable services for them is a challenge, for while it is obviously an advantage to have many services to choose from, there is a need to enable methods to find the most appropriate ones. Recommendations will be one of these mechanisms that will permit users to be aware of items (i.e. services, specifically for SOA4All) that can be helpful for them, given their past behaviour within the platform.

The Recommender System is the key component into providing useful recommendations for users. These recommendations actually take place in the Consumption Platform, thus there is a strong relation of the RS component with that platform, which will query the RS for relevant recommendations.

It is worth noting that the recommendations will be useful not only for helping users to find relevant services by itself, but also because the way these recommendations take place in an active mode will be able to increase the user experience within the platform, hence making them more bound to engage within SOA4All.

It is also important to point out that the relation of the RS with the Consumption Platform is not only in one direction (the outcomes of the RS benefiting the platform) but in the opposite direction as well, for the RS will need to know about the interactions of users within the platform. Figure 1 depicts this two-way relationship, which is then detailed in Section 1.b:

- Consumption Platform → RS: The actions of the users within the Consumption Platform are tracked. These logs track the user behaviour and are gathered thanks to the Logging/Auditing service, which is part of the Management Services made available by T2.4 Infrastructure Services. In turn, the RDF logs are stored through the T2.4 Storage Services. Notice that apart from the general logs of actions, the Feedback Framework (part of the T2.1 Provisioning Platform) will take care of RDF information which is also relevant for the RS (reviews in the form of ratings and comments).
- RS → Consumption Platform: The computations made by the RS are fed back into the Consumption Platform, thanks to suitable calls to the RS API.

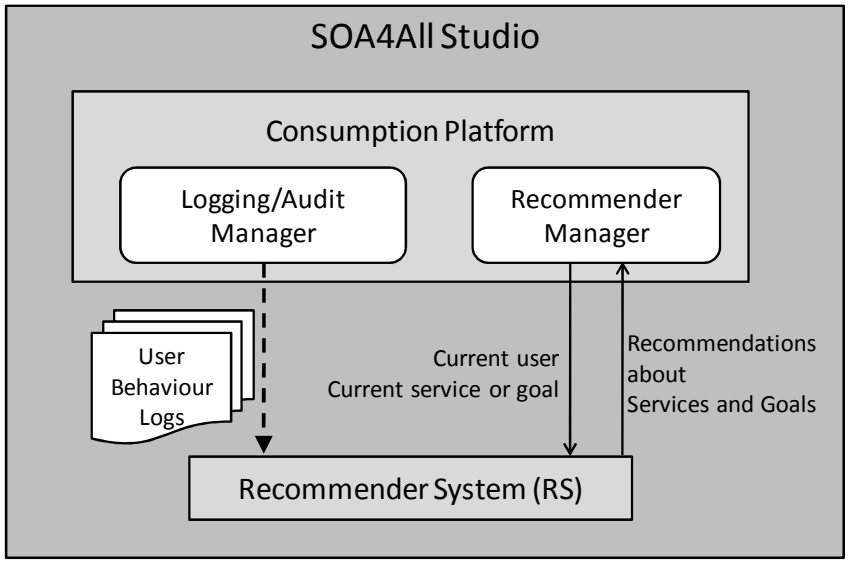

Figure 1: Relation between Consumption Platform and the Recommender System in the SOA4All Studio

Both the RS (T2.7) and the Consumption Platform (T2.2) are components of the SOA4All Studio, and as such it is important to note that the type of communication that happens between them can be achieved by direct Java calls, as depicted in Figure 2.

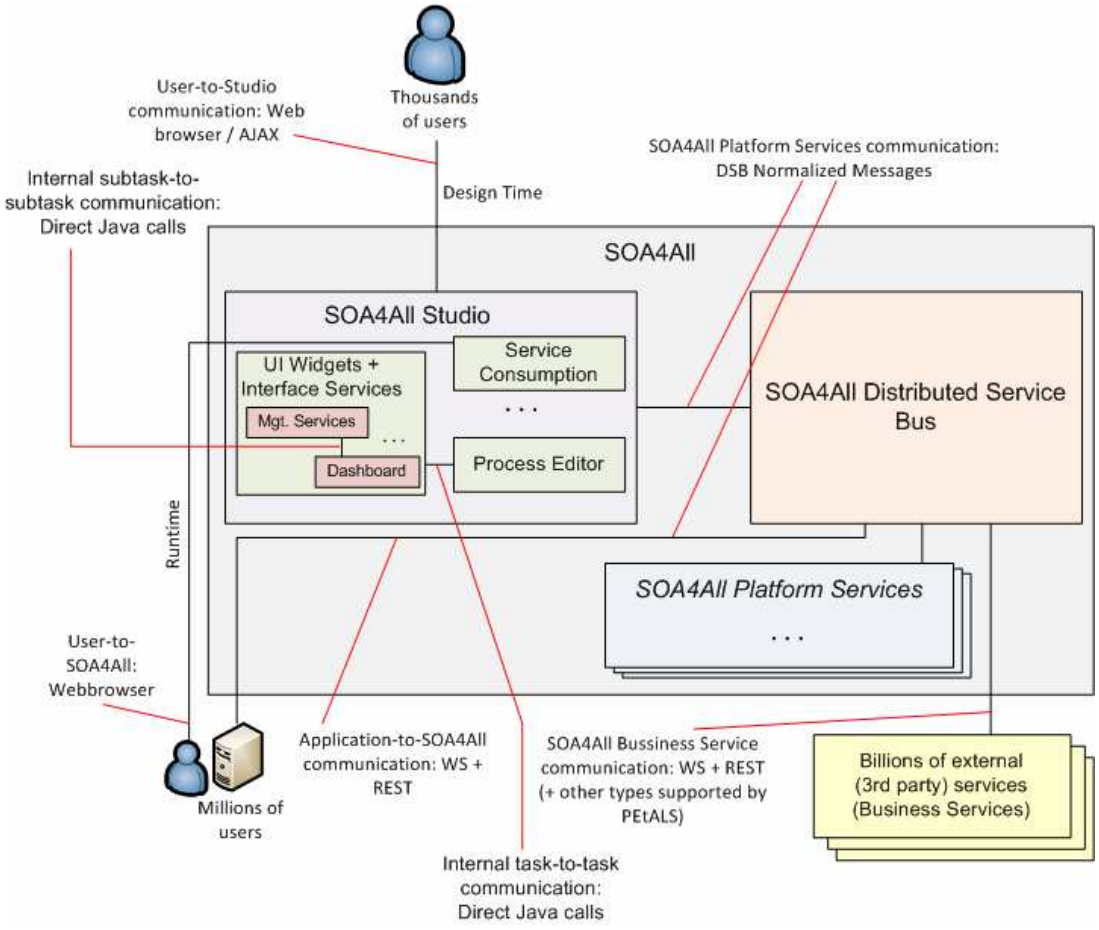

Figure 2: Types of communication in SOA4All

2.2 Relation with the Use Cases

The three use cases can highlight the role of the RS in different ways. While it is not a must for any of them to have recommendations, we will see that they will be quite beneficial and improve each of them.

End-user Integrated Enterprise Service Delivery Platform

The scenario depicted by WP7 implies users repeating similar tasks, who will be likely encountering themselves dealing with similar problems, which will require similar solutions. The RS will be able to understand what the most appropriate choices will be for a given user in each moment, based on previous interactions of similar users who had the same objectives.

The end-users of this use case will benefit from these recommendations, enhancing the ease-of-use in their interaction with SOA4All technology.

W21C BT Infrastructure

End-users interacting with the enhanced Ribbit platform will benefit from recommendations based on the interaction of previous users with the platform. The RS will be able to highlight the most relevant services at a given time for users, based on previous interactions of similar users.

For BT, the fact that users are able to discover interesting services, with the platform suggesting them in an active manner, will be very important in order to engage users within the platform, for they will be presented with interesting options that the users might not be aware of.

C2C Service eCommerce

The RS is capable of automatically recognizing users' interests by analyzing their behaviour within the Consumption Platform, hence suggesting services that users with a similar profile rated in a positive way.

Again, the fact that the RS acts in an active manner will imply that the experience of the users will be improved, as it will be easy for them to get involved into the use of SOA4All technology.

3. Installation and Configuration

Recommender System prototype software is included in D2.7.1-P.zip file (where this document is also included).

Requirements for a complete installation of the prototype are:

- **Java JDK 1.5** or greater.
- **MySQL Server 5.1** or greater. For installation instruction, see http://dev.mysql.com/doc/refman/5.1/en/installing.html.

The prototype installation procedure is as follows:

- Unzip D2.7.1-P in a directory, hereafter referred as %RS_HOME%.
  - %RS_HOME% contains this document and one Eclipse projects, named Recommender System, which contains the Recommender System prototype.
- Set up the databases which will be used by the Recommender System to store intermediate results, by loading the file containing SQL instructions into the DBMS console. Actually, there are two internal databases: one is used by the batch process to analyze the logs and assess the correlation values between users and services, while the second one is used to make recommendations on-demand. To set up the databases, open the MySQL Command Line Client and, after the insertion of username and password, type the following instruction and then press the return key:

  ```
  source %RS_HOME%/src/main/resources/config/RSDB.sql;
  ```

  This command creates two databases and their internal tables, as well as two users with appropriate rights which will be used by the RS software to access to the database. It assigns to the batch-time database and the on-line database respectively the names `UserHistory` and `CorrelationMatrix`. Moreover, it creates only one user to use both databases locally (i.e. on the `localhost`): the username is `rs` and the password is `rs4SOA4All`. If you prefer to use other names for the databases or other usernames/passwords (or you prefer to use a remote database), open the file and change them before launching MySQL. Please, take note of any change you made because it has to be aligned with the configuration file described below.
- The software can be imported in Eclipse for further development, analysis, debugging, testing, etc. within this IDE. The procedure is as follows:
  - Select in Eclipse the menu File/import…, General/Existing Projects into workspace. Next.
  - Check Select Root Directory, click on Browse and locate %RS_HOME%. Select Recommender System project and accept. Finish.

Before starting to run the prototype, it has to be configured appropriately. The configuration can be done by editing the `rs-config.properties` file included in the config directory. Such file contains a list of parameters, each one represented in the form of a pair: `name=value`.

Among others, there are two groups of parameters that refer to two different databases: it is possible to easily identify them because their names begin with the name of the databases, i.e. their names begin respectively with the terms `UserHistory` and `CorrelationMatrix`.

The parameters corresponding to the databases are[1]:

- *database-name*.driver, which refers to the driver that allows to access to the database
- *database-name*.protocol, which refers to the protocol used to interact with the driver
- *database-name*.ServerName, which refers to the address where the server is located (if you have set up the databases with the standard SQL script file, it should be equal to `localhost`)
- *database-name*.Port, which refers to the port number where the database can be reached
- *database-name*.DatabaseName, which refers to the name of the database (if you have set up the databases with the standard SQL script file, it should be equal to either `UserHistory` or `CorrelationMatrix`)
- *database-name*.username, which refers to the username to be used to access to the database (if you have set up the databases with the standard SQL script file, it should be equal to `rs`)
- *database-name*.password, which refers to the password to be used to access to the database (if you have set up the databases with the standard SQL script file, it should be equal to `rs4SOA4All`)

An example of such parameters for the User History can be the following one, where a MySQL database running on the local machine has been used.

```
UserHistory.driver=com.mysql.jdbc.Driver
UserHistory.protocol=jdbc:mysql:
UserHistory.ServerName=localhost
UserHistory.Port=3306
UserHistory.DatabaseName=UserHistory
UserHistory.username=rs
UserHistory.password=rs4SOA4All
```

Other parameters that need to be tuned are the ones representing the actions that are logged by the system in which the RS will be deployed and that the RS has to analyze. As for the databases' parameters, they can be easily located because their names start with the term `Action`.

The RS is able to recognize a predefined set of actions. For each of them, one parameter has to be set: the URI of the action. This is needed because the RS has to be able to internally recognize the different actions and calculate the strength of the relation between users and services by applying the weights defined during configuration.

For example, concerning the action where a user tags a service (i.e. the action named ItemTagging), their parameters can be set as follows:

[1] In this list, *database-name* is just a placeholder that stands for the real names of the databases. In this case, it can be replaced with both `UserHistory` and `CorrelationMatrix`.

```
Action.ItemTagging=http://www.SOA4All.eu/ontologies/logging#
                   ItemTagging
```

In the RS configuration file, other parameters appear, but it is not necessary to change them because they are used by internal processes of the RS to calculate intermediate results. For example, for every action there is an additional parameter whose name ends with the term `weight`: it represents the importance of the action in establishing a strong relation between the user that did the action and the service that is the target of the action. For example, the following line means that the ItemTagging action has an importance equals to 6:

```
Action.ItemTagging.Weight=6
```

Obviously, the importance of an action depends on the maximum and minimum values, so the weight of an action has to be compared with the values of the other action weights in order to understand its importance in establishing a relation between a user and a service. Apart from that, a negative weight decreases the relation (i.e. whenever such action occurs, it means that the user is not interested in the service), while a positive one increases it (i.e. the user is interested in the service).

Starting from the action weights, the RS assesses the correlation values between users and services, which represent the strength of their relations. To do this, it needs two additional parameters storing respectively the minimum and the maximum values allowed for such relations (i.e. allowed in the correlation matrix). The example below shows that a minimum value of 1 and a maximum value of 10 have been set.

```
CorrelationValueMin=1
CorrelationValueMax=10
```

An additional parameter that can be tuned stores the average rating number that a user can use to vote an item (e.g. a service). It is used internally by the RS. The example below sets this parameter to 3.

```
Average.Rating=3
```

4. Software description

The Recommender System is totally integrated in the SOA4All Consumption Platform in a transparent way: the final user of the platform will receive simple and intuitive indications about potentially useful services, without having to deal with any configuration or data request. The system will study in a silent fashion the user actions and interests, providing him the most suitable suggestions in the most appropriate way.

4.1 Sequence diagrams

In this sub-section we report a couple of sequence diagrams useful to understand the relationships existing between the consumption platform and the recommendation system.

For a better clarification of the diagrams, some squares ( ) are displayed on the object lifelines to introduce what kind of action the specified actor is performing.

The first scenario is probably the most common one: after a user logs into the SOA4All consumption platform, he usually starts looking for some sort of services (specifying some search keywords or selecting a particular class in the service taxonomy).

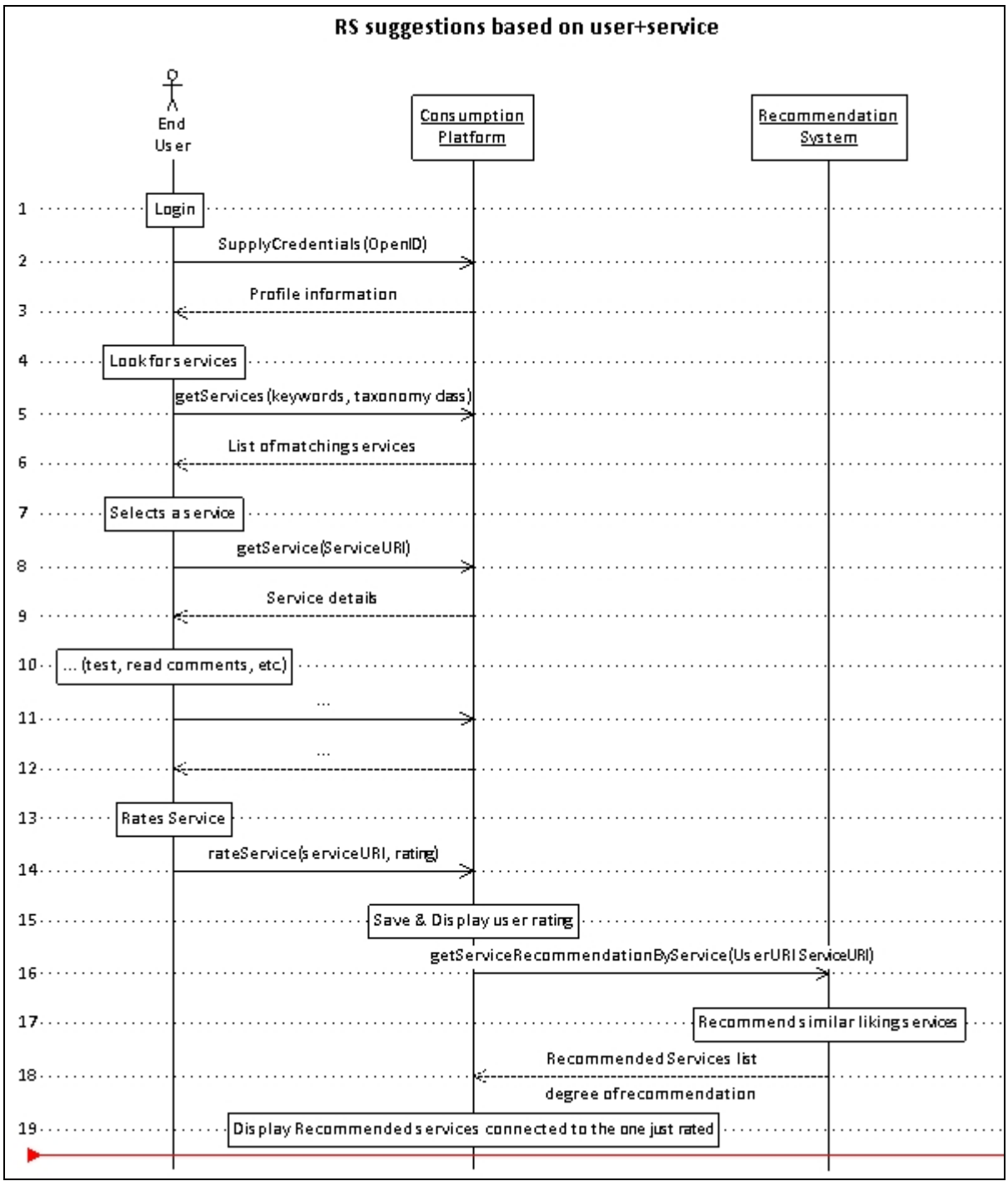

Figure 3 - New user recommendation: sequence diagram

The end user can select one of the services coming out from his research and begins to interact with it, looking at its details, at its comments and tags and maybe testing it through the CP interface or rating the service with a good mark.

In this case, when the user interacts with a specific service, the Consumption Platform contacts the Recommender System to look if it can suggest some other services to the user, by taking into account the current user and the specific service with which he is interacting. Figure 3 shows a possible instance of the scenario described so far.

Another kind of recommendation is provided whenever a user logs-in (see Figure 4).

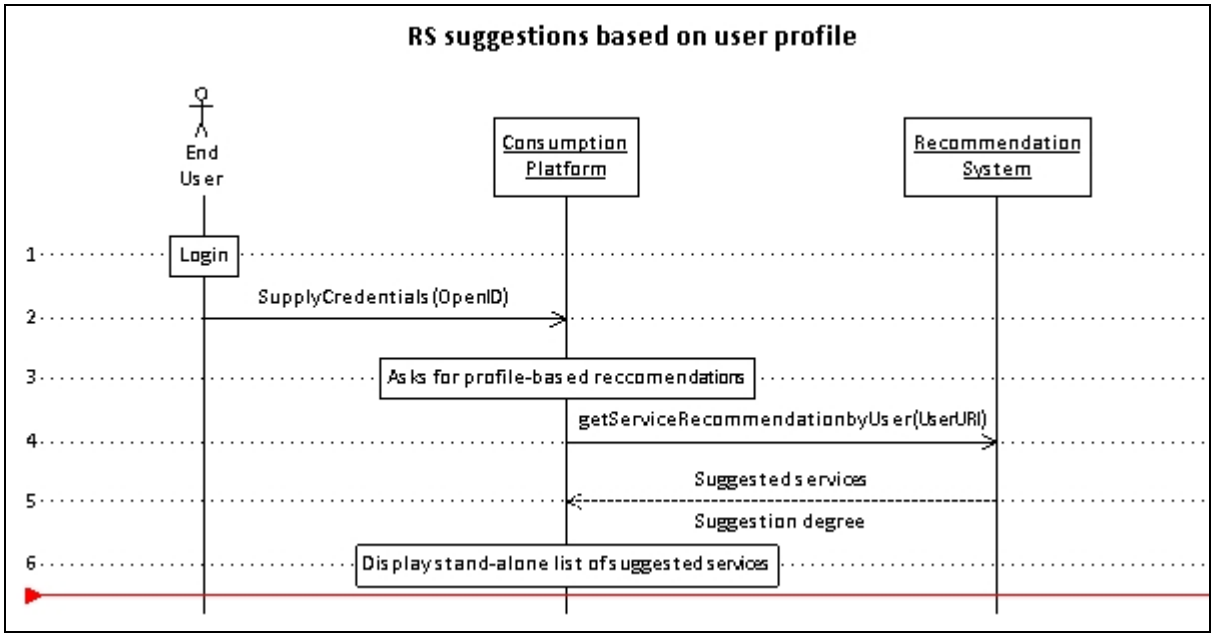

Figure 4 - Profile-based recommendation: sequence diagram

As soon as the user logs-into the Consumption Platform, this component contacts the Recommender System to check whether any suggestion is available for the current user. The RS loads it's off-line computed data and returns back to the CP a list of potentially interested services for the specified user, together with their degree of possible interest.

It should be underlined that these suggestions are not produced at the moment the RS is queried, but they are the result of a batch cycling computation over the whole amount of historical logs produced by the user.

To better clarify the batch process, it could be useful to show a simple sequence diagram of this phase too.

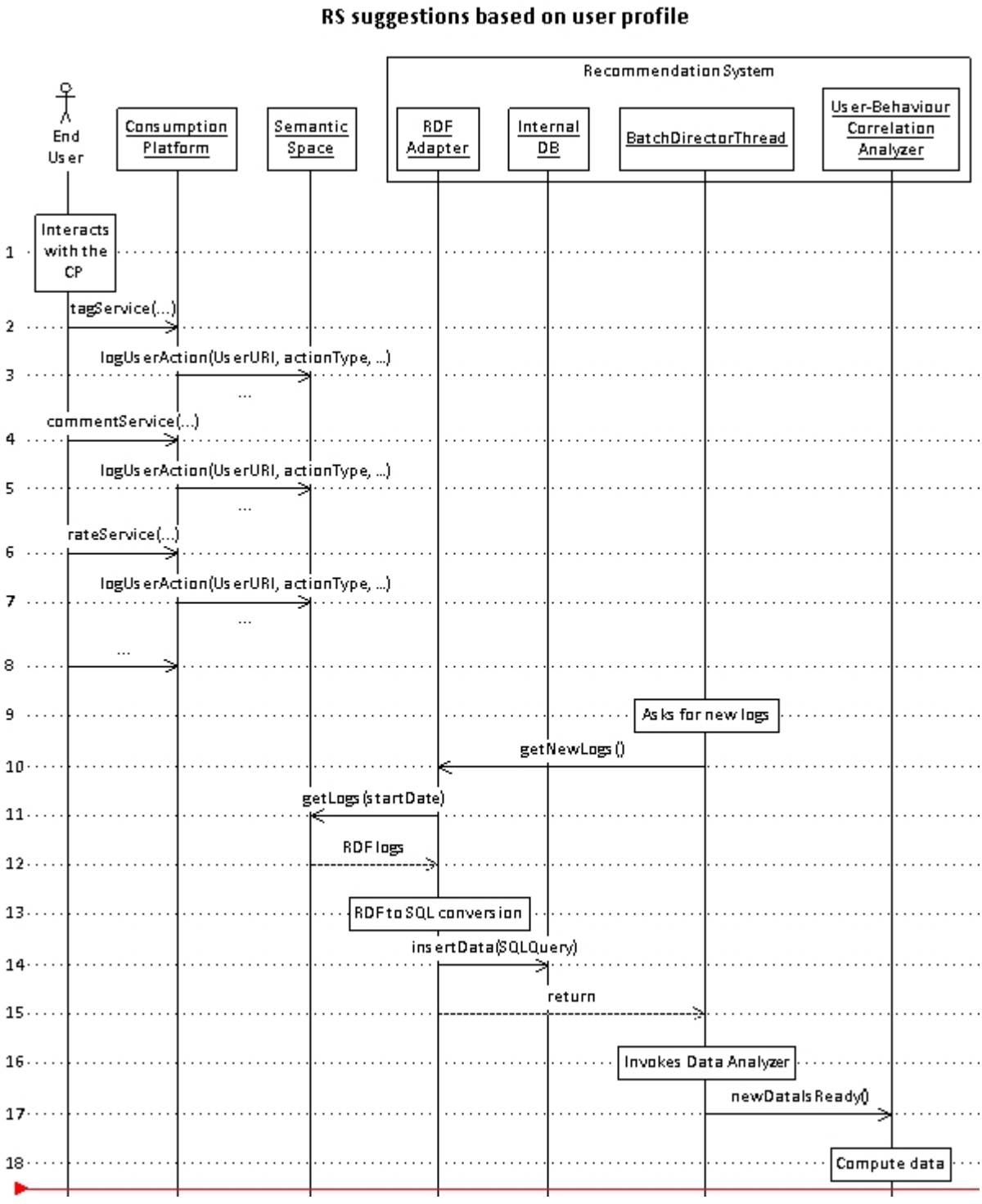

Figure 5 - Batch computation: sequence diagram

The logged users, interacting with the platform, automatically produce logs that are silently stored in the SOA4All semantic space. Each periodic delta time (configurable in the RS) the RS contacts the Semantic Space APIs to retrieve all the new logs produced by the users actions in the platform since last batch computing. The logs are returned as RDF triples.

The RDF Adapter, which analyses them, fetches these triples and, for each one of the logs, creates an appropriate new entry in the RS internal relational database.

Now these data can be processed and computed by the User-Behaviour Correlation Analyzer to discover new suggestions to be supplied to the Consumption Platform in real-time. Such analyzer works by updating the internal correlation matrix that defines the relations between users and items. To do this update, the analyzer goes through all the actions performed by each user and apply the weights on the specific cells identified by the user and the item.

4.2 First version of the GUI

As described before, the Recommender System will be integrated in the Consumption Platform, so the Graphical User Interface has been studied to be totally encapsulated inside the CP.

Depending on the kind of recommendation, suggested services will be shown in different places of the platform. Suggestions based on the current service will be displayed in the bottom area of the service details portlet, in an *ad hoc* section called “Suggestions”. If no suggestions are present (this may happen if no one has never user this service), this area of the portlet will be collapsed and not selectable, as shown in the following picture.

Figure 6 - Service portlet with "Suggestion" section not enabled

If suggestions are available, the “suggestion area” is activated and the user has the possibility to expand it as he does for the other sections of the portlet.

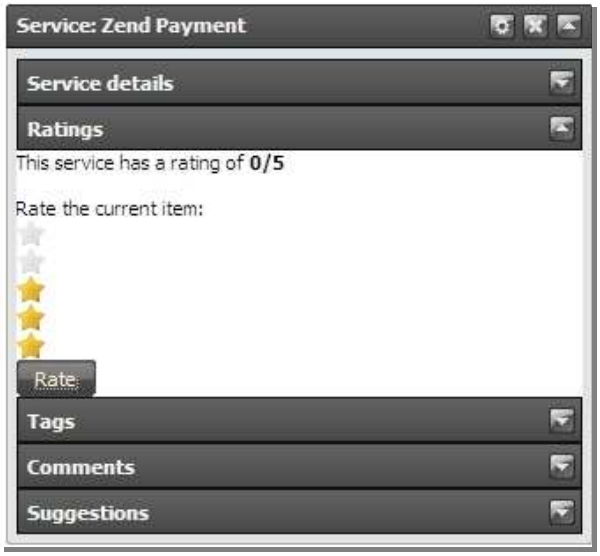

Figure 7 - Service portlet with "Suggestion" section enabled

In this area there is a textual HTML table displaying the suggested services names and the percentage of potential interest the user can have about each of them.

Figure 8 - Service portlet with suggested services table displayed

Clicking on a service name in this table, a new portlet with all the service details will open in the Consumption Platform desktop. Such in a way the user will be able to have more information about the suggested service and interact with it.

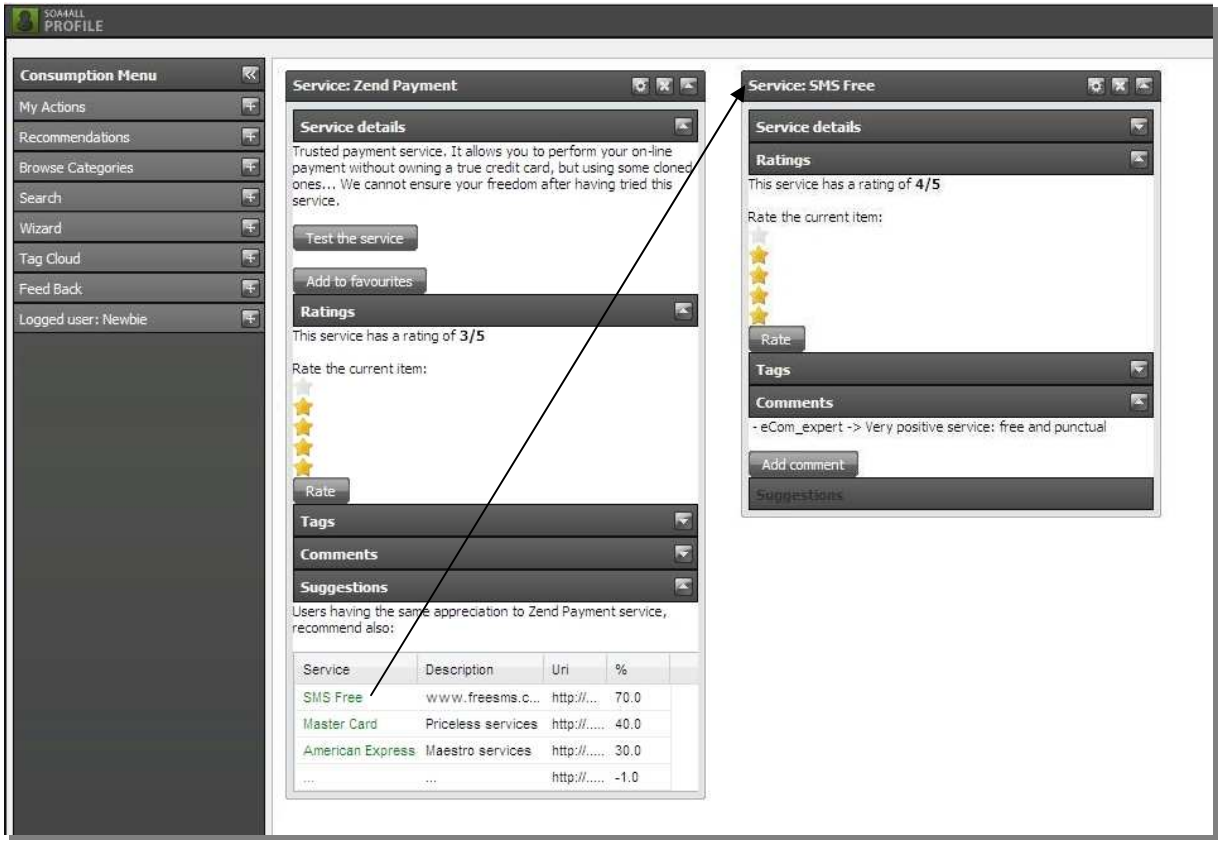

Figure 9 - Selection of a suggested service

In the case the user is logged in and the RS had time to analyze his past history (this usually occurs within 24 hours because batch time operations are performed during night-time), the CP will show recommendations not bound to the current viewed service. In the main menu, positioned on the left of the dashboard, there is a specific section to display suggested services, called “Recommendations”. In case of new suggestions the title in the menu changes colour, becoming red, to attract the visitor’s attention.

Figure 10 - Recommendation area in CP main menu.

“No suggestion” vs. “suggestions available” appearance

The user can expand the “Recommendations” section and, if suggestions are present, they are displayed in a HTML table as for the previous example. In this case, services are displayed with different shades of colour: the darker the colour, the more confidence the recommendation has. Moving the mouse on a suggested service, a tool tip is displayed showing the average rating of the specific service and its suggestion confidence, as it is shown in the following figure.

As in the case above, a click on an item of the table will open the service portlet where all the service information are displayed. In the main details section a brief explanation of the reason of the suggestion and the related percentage degree are shown.

Figure 11 - Recommended services by user profile

This area, displaying user profile-based recommendations, can be updated automatically every time the RS concludes a batch computation cycle, providing new suggestions for the current user, or every time the user logs in.

4.3 Package description

This section describes the Recommender System projects through some Eclipse IDE snapshots.

As described in the installation section, D2.7.1-P.zip creates project directories after being unzipped. Next picture shows the content of the project in the Eclipse Package Explorer.

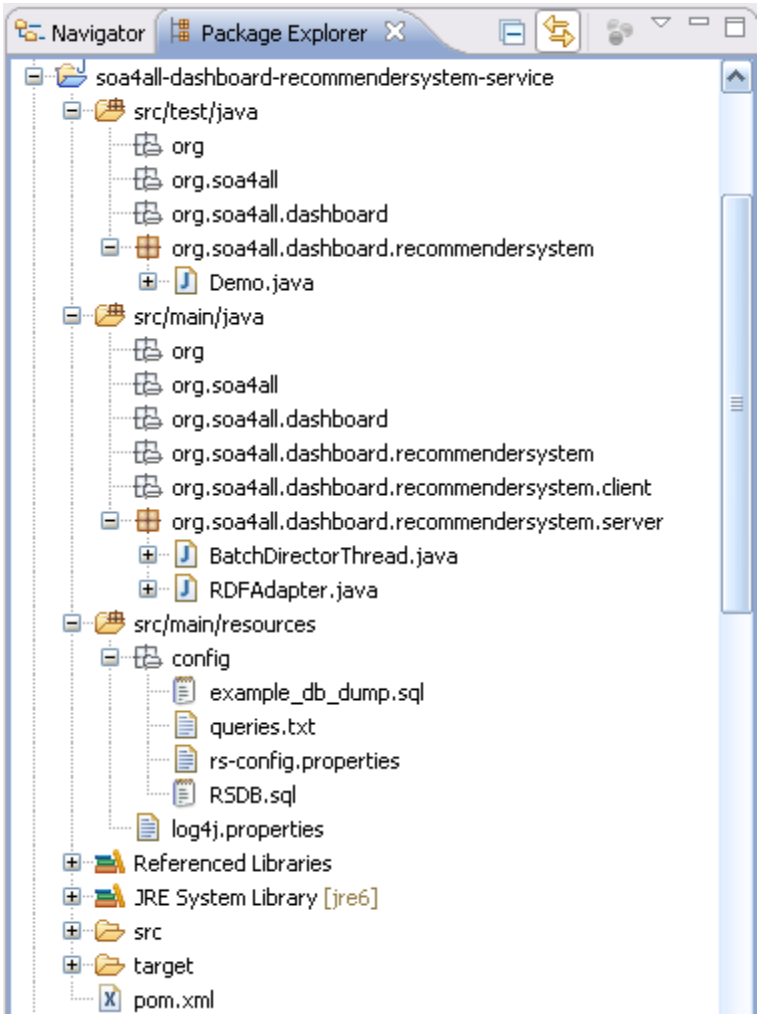

Figure 12 Recommender System project content

The project is organized as follows:

- src/test/java/ contains the class implementing the available test case. There are scripts available in the scripts directory within the same purpose.
- src/main/java/ contains the source code of the classes that make use of the RS API.
- src/main/resources contains the configuration files. In particular, the file named `log4j.properties` defines the level of the messages printed in the console during the execution of the RS (for debug or general information). The subfolder named config includes the files used to customize the behavior of the RS (e.g., they enumerate the actions available in the logs as well as their weights, the parameter to access to the databases, and so on).
- target/ is the folder where the compiler puts the output files.
- `pom.xml` contains the description of the project as well as the references to the library used by the project.

Among the others, the pom.xml declares a special dependency towards the library named `RS-0.1.jar`. In this library we put the core of the Recommender System developed by CEFRIEL.

5. Description of tests execution

In order to verify that the RS component has been correctly deployed, it is possible to run a simple test included in the package.

The test consists of inserting sample data in the User-Service correlation matrix and running the RS to get some recommendations. The data represent the connections between four fake users and four services and they have been derived by simulating the behaviour of the users within the SOA4All Studio.

The users that are present in the sample data are identified through their URIs, which are:

- mailto:user1@example.com
- mailto:user2@example.com
- mailto:user3@example.com
- mailto:user4@example.com

On the other side, the URIs of the four services included in the sample are:

- http://ws.acrosscommunications.com/ICQ.asmx
- http://www.arcwebservices.com/services/v2006/WirelessLocation
- http://www.infocountry.co.za/za/default/webservice/SMSService.asmx
- http://www.webservicex.com/sendsmsworld.asmx

To set up the environment for testing the RS, take the file named `example_db_dump.sql` under the config directory and load it into MySQL with the following procedure:

- Open the MySQL Command Line Client and insert your username and password
- Type the following instruction (you have to replace `%RS_HOME%` with the absolute path to the installation folder of the RS) and then press the return key:

```
source %RS_HOME%/src/main/resources/config/example_db_dump.sql;
```

With the execution of the instruction above, you insert some data into the RS databases (both UserHistory and CorrelationMatrix). So now you can run the RS run-time component to get some recommendations. To do that, we created a special application that asks recommendations to the run-time component of the RS.

- Open the Eclipse project of the RS (this step is optional because you can use the command line, if you prefer).
- Run the class `Demo.java` in the directory `src/test/java`.

The application shows you a menu where, besides the option to quit the program, you can choose among three options:

1. Get recommendations given a user
2. Get recommendations given a service
3. Get recommendations given a user and a service

At this point, for example, after choosing option 1., you are asked to insert the URI of a user to get a list of recommended services for that user. So, if you insert

mailto:user2@example.com you get the following result[2]:

1. [10.0] http://www.webservicex.com/sendsmsworld.asmx
2. [3.528894] http://www.infocountry.co.za/za/default/webservice/SMSService.asmx

If you got such results without any error, it means that the RS has been installed successfully.

[2] The values given between square brackets give an indication about the relevance of the recommendations. Higher values denote stronger recommendations.

Annex A. Design of the Recommender System in SOA4All

This annex describes the contribution of the activities performed for the development of the Recommender System [3].

1. The Recommender System in SOA4All

The Recommender System (RS) in SOA4All aims to provide users with recommendations about services (the recommended items) that could be of their interested. This functionality acts as an additional feature to support users in discovering services that meet their needs.

This chapter describes the RS in SOA4All from a functional perspective, starting with the approach adopted, its placement within the Consumption Platform, the API provided and the necessary inputs it requires.

a. Approach adopted in SOA4All

There are two main and distinct approaches for recommendation: content-based and collaborative filtering. Within the scope of SOA4All, we aim to exploit users' similarities in interacting with the SOA4All Studio to provide users with those services used by similar users. This intention relies on collaborative-filtering techniques.

Given a user, a collaborative filtering recommender system suggests him the items that he did not already seen while other similar users (i.e. users that saw more or less the same set of items with similar preferences) appreciated a lot. This is the source of probably the main positive effect of the collaborative filtering approach, which is the increasing of serendipity, in the sense that such a system suggests completely different items with respect to the ones the user has already seen in the past. This kind of behaviour is usually appreciated, especially in domains where the user is more inclined to use an unseen item (e.g. music, books, movies). However, this is not really the case for Web services, where usually a user who is looking for a specific type of service does not take into account other services of a quite different type. In addition, collaborative filtering approaches have the drawback of requiring to gather and to analyze a considerable set of user's interactions before being able to infer the implicit similarities among users and to provide recommendations. This drawback is also called as "cold start" problem.

On the other hand, content-based approaches reduce such drawback by analyzing the content of the items in order to understand the similarities among items (i.e. services). Within the scope of SOA4All, services will be semantically described, so that semantic relations can be exploited to evaluate the similarities among them in order to provide users with suggestions related to services that are semantically similar to the ones previously used.

For this reason, the RS in SOA4All will be initially based upon collaborative-filtering techniques (by M18) and, in a following phase (by M30), a hybrid approach will be adopted by considering semantic-based techniques over the semantic descriptions of services. This will reduce the impact of the cold start problem of the pure collaborative approach.

The following sections describe the RS designed for M18 thus considering only the collaborative filtering approach.

[3] Since this part is not part of the official project template for prototype deliverable, we decided to include this part as a annex.

b. The RS in the Consumption Platform

According to the collaborative filtering approach, the RS has to analyze user behaviour in interacting with the Consumption Platform. Such behaviour is tracked in specific logs generated by the consumption platform and stored inside the Semantic Space.

In Figure 13 we report the relations between the RS and the Consumption Platform.

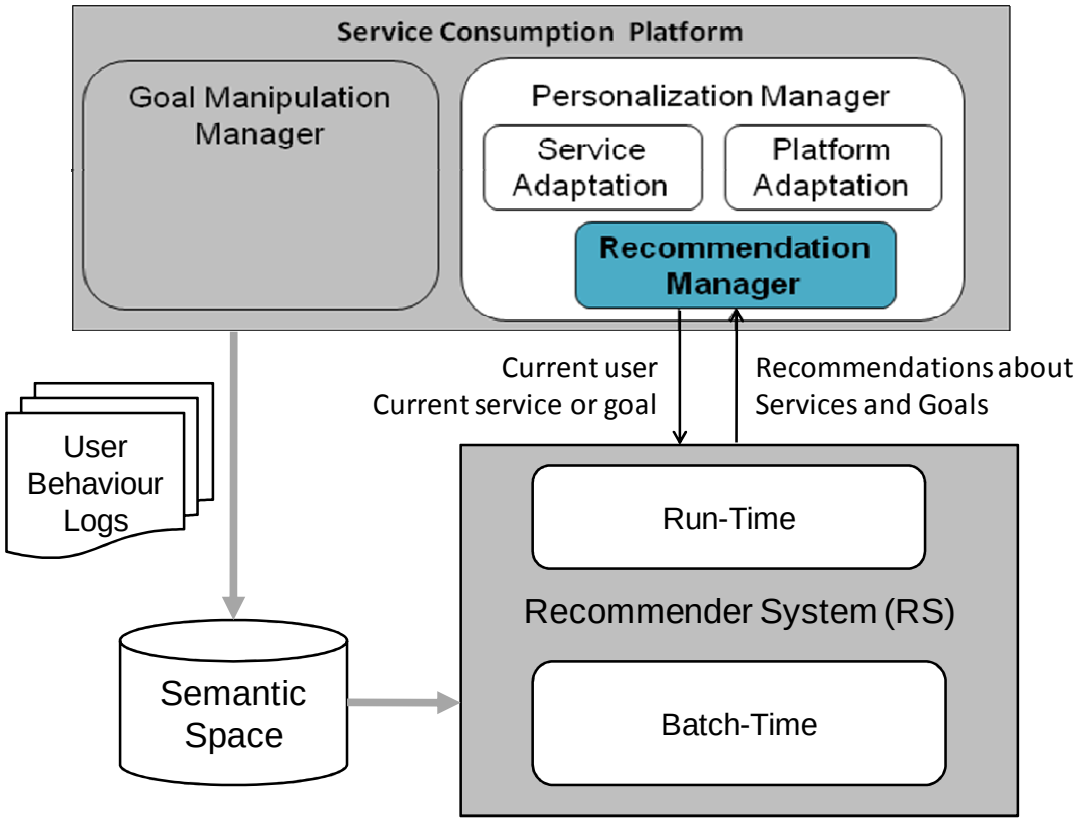

Figure 13: The Recommender System and the Consumption Platform

The SOA4All Studio logs generated by the platform are analyzed during batch-time by a specific component inside the RS. After analyzing the user behaviour, the RS is able to provide recommendations at run-time.

The run-time component receives some context information such as the user currently logged in and the service currently inspected. Based on such information and on the previously analysis, the RS provides recommendations about services to the user. The recommendation manager acts as a client for the RS and bridges the requests inside the Consumption Platform with the functionalities provided by the RS.

The following paragraph describes the considered actions performed by users and the next paragraph reports on the API exposed at run-time to the Consumption Platform.

c. Actions and RDF(s) for modelling User Behaviour

User's behaviour is tracked in specific logs stored inside the Semantic Space. Such behaviour is represented as the chronological list of all the actions performed by each specific user. The following table summarizes the actions and their relative parameters analyzed by the RS in order to infer user similarities.

Table 1: Actions and Parameters

Action Type	Additional Parameters
LoginAction	UserURI
LogoutAction	
ItemSelection	ItemURI
ItemTagging	ItemURI, Tag
ItemEditing	ItemURI
ItemCategorization	ItemURI, Category
Review	Rating, Text
ItemInvocation	ItemURI

Since the logs are managed inside the Semantic Space, an RDF representation is needed. Annex B reports the RDF(S) in N3 format used to model the RDF entries in the Semantic Space, while Annex C contains some examples of entries defined for testing purposes.

The actions of the previous table are represented by specific concepts that inherit from the generic Action concept. Each action performed by a user is also related to a LogEntry that contains information about the datetime of the action and the session ID. To represent datetimes, we reuse the Instant concept of the Time ontology[4].

Consequent actions performed by the same users are bound together using the webapp session ID. If the user perform a login action, all the actions within the same session will be reconducted to such user. Anyway, to correlate actions of users among different webapp sessions without requiring the user to login, the session ID can be used as a persistent session ID by storing the webapp session ID in a persistent cookie on the user browser.

Since users can provide ratings and comments about services, the RDF(s) is also designed to reuse the Revyu schema[5]. There is a specific action named ItemReview that links to the Review concept of the Revyu schema.

During the setup phase, the RS has to be configured in order both to understand all these actions and to be able to manage them with appropriate levels of importance. Therefore, a weight is associated to each action. Higher positive weights are associated to actions giving a clear evidence of the high appreciation of the item involved in the action, while lower positive weights are associated to actions where the appreciation is not so evident or is lower. On the other hand, the actions giving a clear evidence of the rejection of the item are associated with negative weights. The values associated to weights are defined empirically by assuming the implicit importance that a user may give to an item (service) in function of the specific action performed. The following table reports an initial estimation of the weights

[4] The Time ontology is described at http://www.w3.org/TR/owl-time

[5] The specification of the Revyu schema are accessible at http://danja.talis.com/xmlns/rev_2007-11-09/index.html

associated with some of the actions.

Table 2: Actions and Weights

Action	Approximate Weight (-10 min, 10 max)
Review and Rate High and Comment	10
Review and Rate High	8
Item Categorization	7
Item Tagging	6
Item Editing	5
Item Invocation	4
Item Selection	1
Review and Rate Low and comment	-5
Review and Rate Low	-10

d. RS API

At run-time, the RS provides the Consumption Platform with recommendations about services that are related to the current user and/or the current service. The communication between the CP and the RS is performed via Java API.

3 dedicated operations are provided by the RS for recommending services. The following listing reports the three methods for getting recommendations about services.

```
public List<RecommendedItem> getServiceRecommendationByUser
            (URI userURI, int howMany);

public List<RecommendedItem> getServiceRecommendationByService
            (URI serviceURI, int howMany);

public List<RecommendedItem> getServiceRecommendationByUserAndService
            (URI userURI, URI serviceURI, int howMany);
```

The first method, getServiceRecommendationByUser, aims to suggest services to the user (identified with his userURI). The implementation behind this operation exploits user similarities to recommend those services used by similar users.

The second method, getServiceRecommendationByService, aims to recommend services that are similar to the current service (identified with its serviceURI). This implementation returns the other services inside the clusters of the current service. Such service clusters are derived by aggregating the services used by similar users.

The third method, getServiceRecommendationByUserAndService, returns those services that are relevant with regards to the current user and the current service. This is an hybrid implementation of the previous two methods that intersect the clusters of similar users with

the services used by such users.

Each of the three methods returns a list of objects (i.e. the recommended items), whose contents include only two fields: the URI of the recommended item and the value of confidence (or strength) computed for the recommendation of the item.

2. Architecture

This section describes the general architecture of the RS in SOA4All and provides some details about its main internal components. Figure 14 describes this architecture emphasizing the role of the batch-time component that analyzes user behaviour and the run-time component that provides recommendations to the Consumption Platform.

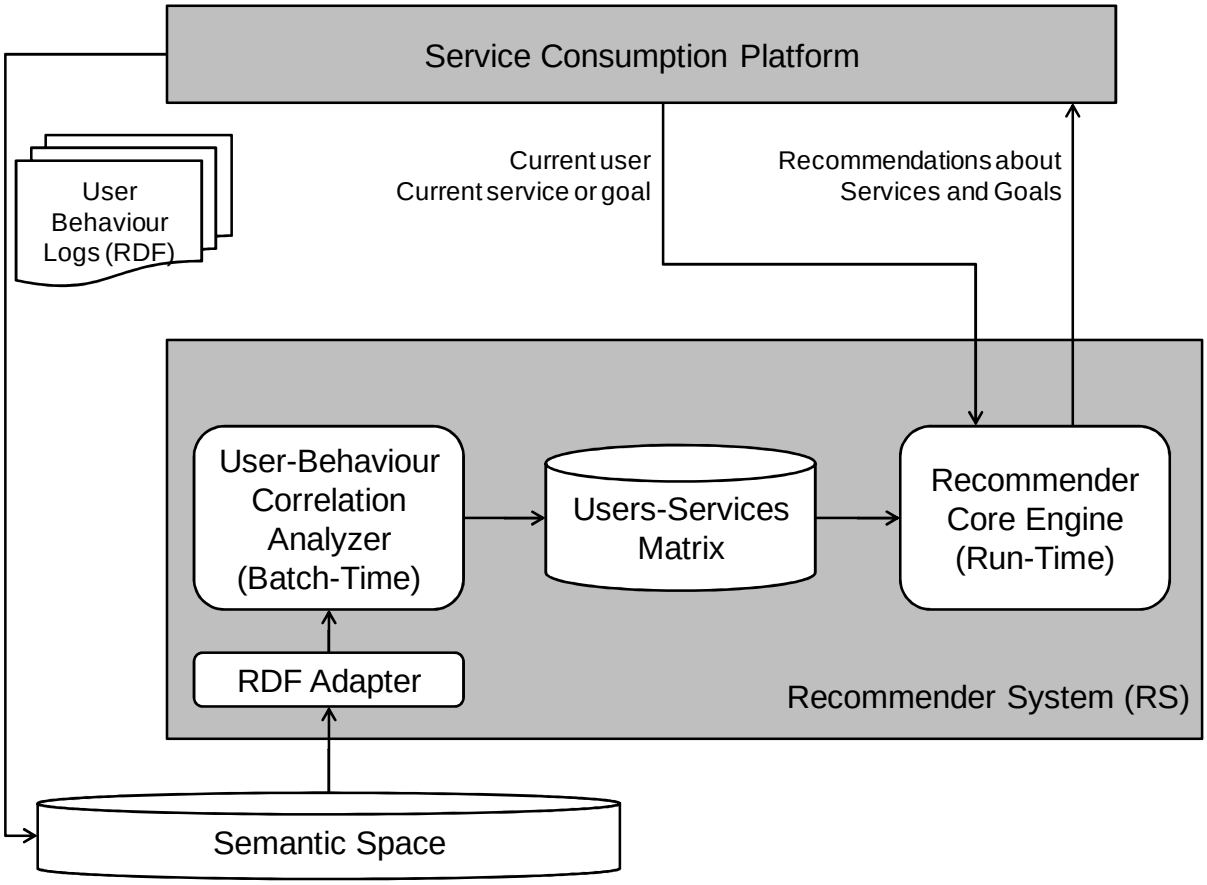

Figure 14: The general architecture of the Recommender System

The user behaviour is represented as RDF log entries within the Semantic Space. The RDF Adapter is in charge of getting log entries from the Semantic Space and extracting their content that is passed to the User-Behaviour Correlation Analyzer.

Such analyzer analyzes the data and fill the matrix for correlating users with services.

During run-time, the Recommender Core Engine receives requests from the Consumption Platform and, using the analyzed data, provides recommendations.

a. RDF Adapter for Parsing User Actions

The RDF Adapter is responsible for getting log data produced by the SOA4All Studio and extracting information from them. The extracted information is temporarily inserted into an internal structure named *User History*, which represents the history of all user actions done in the past.

In particular, the RDF Adapter is an internal component that converts RDF log instances into specific property objects and then stores them in the Recommender System relational database, following these rules:

- an RDF node is a record of a table;
- an RDF propertyType is the field name (i.e. the name of a column of a table);
- a literal value of an RDF triple is the value of a record field of the table.

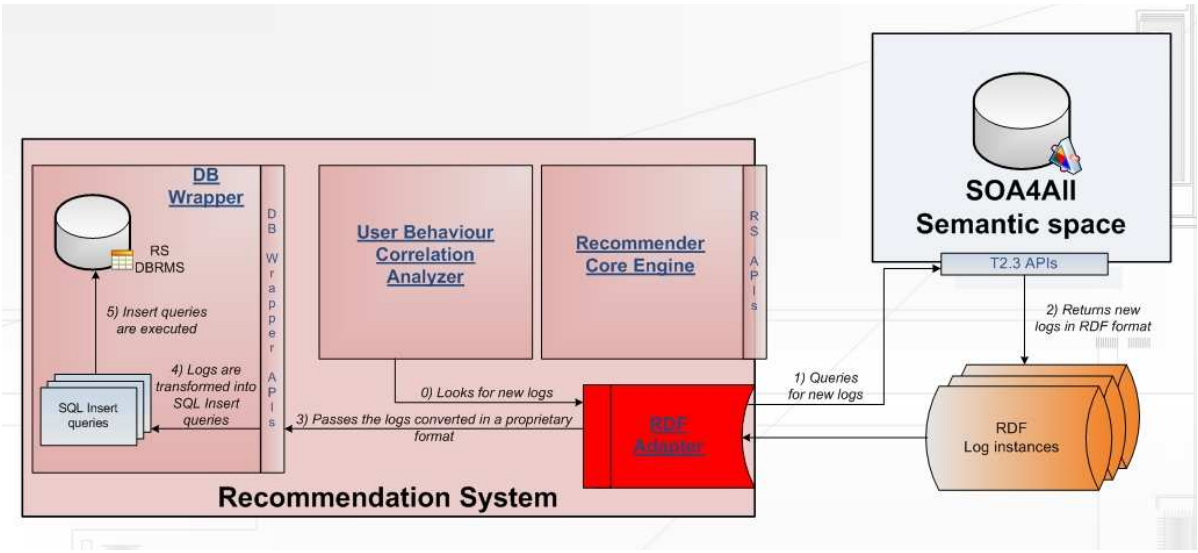

Figure 15 - Data flow from SOA4All studio logs to Recommender System internal database

b. User Behaviour Correlation Analyzer

This component is responsible for estimating the strength of the relations between users and services[6], which means to fill in the matrix that expresses user interests for the services. In other words, it examines the User History in order to produce the User-Service Matrix.

User-Service Correlation Analyzer considers every user separately. Firstly, it collects all actions made by a specific user. Then, for each action, it gets the reference to the service related to that action and updates the cell of the matrix that corresponds to the intersection between the row of the user and the column of the service by adding the action weight.

Formally, the User-Service Correlation Analyzer runs the algorithm shown below, which is described using a pseudo-code. *R* represents the User-Service Matrix, which at the beginning is initialized with null values. *R(u, s)* represents the cell of the matrix *R* corresponding to user *u* and service *s*. *max(u)* and *min(u)* are respectively the maximum and minimum values in the relations between user *u* and all services, while *MAX* and *MIN* are respectively the maximum and minimum value used to express the strength of any relations between users and services in the matrix *R*. Given an action *a*, *a.weight* denotes the weight of the action, *a.timestamp* denotes the instant at which the action occurred, *a.user* identifies the user that performed the action and *a.service* identifies the service related to the action (e.g. if a user rated a service, then the service related to that action is the one rated by the

[6] Hereafter we only refer to services as the items that are recommended, but the all approach can be applied similarly to goals as well. Since the recommendation of goals will be implemented in the second version of this prototype, at this time we avoid citing goals for the sake of readability.

user). *lastExecutionTimestamp* stores the instant of the last execution of this algorithm.

```
for all user u do
      updatedStrengths ← ∅
      minOrMaxChanged ← false
      for all action a ∈ UserHistory such that a.user = u and a.timestamp >
lastExecutionTimestamp do
            s ← a.service

            // Update the value of relation between u and s
            R(u, s) ← R(u, s) + a.weight
            updatedStrengths ← updatedStrengths ∪ {s}

            if R(u, s) > max(u) then
                  max(u) = R(u, s)
                  minOrMaxChanged ← true
            end if
            if R(u, s) < min(u) then
                  min(u) = R(u, s)
                  minOrMaxChanged ← true
            end if
      end for

      // Normalization of the values of the user's relations
      if minOrMaxChanged then
            for all service s do
                  if R(u, s) = NULL then
                        R(u, s) = [R(u, s) - min(u)] / [max(u) - min(u)] ·
                                    (MAX - MIN) + MIN
                  end if
            end for
      else
            for all service s ∈ updatedStrengths do
                  if R(u, s) == NULL then
                        R(u, s) = [R(u, s) - min(u)] / [max(u) - min(u)] ·
                                    (MAX - MIN) + MIN
                  end if
            end for
      end if
end for
```

The first time this algorithm is executed, the User-Service Matrix contains only null values. When the algorithm is executed, for each user it extracts from the User History all actions that the algorithm has not yet taken into consideration. Then, every action implies to update the value of a cell of the matrix: the weight of the action is summed to the value already stored in that cell. Finally, a normalization step is executed to spread the values of a user over a common numerical scale in order to make user profile (i.e. the rows of the matrix) comparable. To do that, it is necessary to keep track of both the maximum and the minimum values of the cells corresponding to the user (i.e. *max(u)* and *min(u)*) and compare them with the maximum and minimum values allowed in the final matrix (i.e. *MAX* and *MIN*).

In the calculation of relation values the action weights are summed, so a value will be very high for the users that performed many actions related to the corresponding service, while a value will be very low for users that performed few actions related to the corresponding

service. Since the users perform different numbers of actions, the values contained in the matrix will vary on different scales. To allow comparison between any users (irrespective of the number of actions they performed), it is necessary to shift the values of every user to a common scale (normalization step).

After the normalization of all values, the values in all the rows of the matrix can be used to make recommendations. However, if the algorithm is executed once more (for example, because the User-Service Matrix needs to be updated taking into account new actions performed by the users within the portal), the action weights cannot be simply summed to the value included in the cells because the value has been normalized while the weights have not.

For this reason, a slightly different version of the User-Service Matrix is maintained as well, where the values contained in it are computed by the algorithm described above with the exception that the normalization step is skipped (we named such matrix *Un-Normalized User-Service Matrix*).

c. Recommender Core Engine

Recommender is responsible for making recommendations. Our implementation is based on Taste[7], an extensible framework that implements many recommendation algorithms available in literature.

As explained before, there are three kinds of recommendations that the RS can return (see paragraph 1.d). They are all based on the well-known concepts involved in the collaborative filtering approach. Indeed, as a first step, the RS compares the rows of the user-service matrix – representing the users' profiles – in order to calculate the similarity between users. The most similar users to the given one form the user's neighbourhood. At run-time, such information is exploited to make recommendation.

The first kind of recommendation provided by the RS is represented by the method named getServiceRecommendationByUser, which takes the reference to a user as input. That is, given a specific user, the profiles of his neighbours are taken into account jointly in order to identify the most appreciated services (i.e. the matrix cells containing the highest values). From that set of services, the ones that the given user has not yet seen are recommended.

Another type of recommendation is the one implemented in the method named getRecommendationsByService, i.e. given a service the RS suggests other services that the users tend to use in a similar way as the given one. In particular, given the input service, the

RS identifies the users that used it and collects all the other services used by them. Then, for each collected service, the RS considers its relations with the users and compares them with the relations that the input service has with the same users. The more such relations are similar, the more the service is recommendable. This recommendation confidence about a generic service *x* is computed with the following formula, which uses the same notation introduced in the previous paragraph and where *s* represents the service given as input.

$$L(s,x) = \frac{\sum_{u : R(u,x)\neq NULL} (MAX - MIN - | R(u,s) - R(u,x) |)}{\sum_{u : R(u,x)\neq NULL} (MAX - MIN)} \cdot 100$$

At the end, the services with the highest recommendation confidence are recommended.

[7] http://taste.sourceforge.net

The last kind of recommendation provided by the RS is implemented by the method named getRecommendationsByUserAndService, which takes a service and a user as input. Firstly, it considers the neighbours of the given user and collects all the services they used except the ones already used by the given user. For each of the collected service, it evaluates how much the service has been used by the identified users similarly as the given service, by computing the following formula where *ū* represents the given user and *sim* is a function that returns a value representing the similarity between two users.

$$L(\bar{u},s,x) = \frac{\sum_{u : R(u,x)\neq NULL} | sim(\bar{u},u) \cdot (MAX - MIN - | R(u,s) - R(u,x) |) |}{\sum_{u : R(u,x)\neq NULL} | sim(\bar{u},u) \cdot (MAX - MIN) |} \cdot 100$$

The services with the highest values are then recommended.

3. Conclusions

So far, we developed a recommender system based on the collaborative filtering theory. Currently, the component provides three methods that retrieve recommendations about services. So, one of the future works is to adjust the system in order to recommend goals as well. The methods that will be provided to get goal recommendations will be similar to the ones already implemented for service recommendations. The following listing reports such three methods.

```
Public List<RecommendedItem> getGoalRecommendationByUser(URI userURI, int howMany);

public List<RecommendedItem> getGoalRecommendationByGoal(URI goalURI, int howMany);

public     List<RecommendedItem>     getGoalRecommendationByUserAndgoal(URI userURI, URI goalURI, int howMany);
```

However, the main contribution we are going to design in the next months is to study a new method that exploits semantic description of services and goals to recommend them based on the semantic similarity of their contents (i.e., a semantic content-based approach), instead of the similarity in their usage (which is produced by the collaborative filtering approach). Then, we will investigate how to combine the two approaches in order to improve the quality of recommendations.

Annex B. The RDF(S) used to model user behaviour

```
@prefix dc:   <http://purl.org/dc/elements/1.1/>.
@prefix time: <http://www.w3.org/2006/time>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rev:  <http://www.purl.org/stuff/rev#>.
@prefix s:    <http://www.SOA4All.eu/log#>.
@prefix vs:   <http://www.w3.org/2003/06/sw-vocab-status/ns#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.

s:LogEntry
        rdfs:comment "LogEntry is the single line of a usual log file";
        a rdfs:Class;
        s:hasDateTime time:Instant;
        s:fromIP rdfs:Literal;
        s:hasPersistentSessionID rdfs:Literal;
        s:hasAction s:Action.

s:Action
        rdfs:comment "Action is a generic class that is subclassed in the
various types of actions";
        a rdfs:Class.

#############################################################
#Here the list of some of the Actions (to be completed)

s:LoginAction # This action is very important because it lets the RS to
correlate different sessions of the same users
        rdfs:subClassOf s:Action;
        s:loggedInUser foaf:Person.

s:LogoutAction
        rdfs:subClassOf s:Action.

s:Item
        rdfs:comment "Item is the generic item managed by SOA4All Studio (a
Service)";
        a rdfs:Class.

s:Service
        rdfs:subClassOf s:Item.

s:Goal  # available in the future version
        rdfs:subClassOf s:Item.

s:ItemSelection
        rdfs:subClassOf s:Action;
        s:selectedItem s:Item.

s:ItemBookmarking
        rdfs:subClassOf s:Action;
        s:bookmarkedItem s:Item.

s:ItemTagging
        rdfs:subClassOf s:Action;
```

```
        s:taggedItem s:Item;
        s:tag rdfs:Literal.

s:ItemEditing
        rdfs:subClassOf s:Action;
        s:editedItem s:Item.

s:ItemCategorization
        rdfs:subClassOf s:Action;
        s:categorizedItem s:Item:
        s:category rdfs:Literal.

s:ItemInvocation
        rdfs:subClassOf s:Action;
        s:invokedItem s:Item.

################################################################################
##################
# Binding with the Revyu schema (from
http://danja.talis.com/xmlns/rev_2007-11-09/index.html)
s:ItemReview
        rdfs:subClassOf s:Action;
        s:hasReview rev:Review.

# The hasReview property can be applied to s:Item
rev:hasReview
        rdfs:domain s:Item.

################################################################################
###############
# Import of the Revyu schema (from http://danja.talis.com/xmlns/rev_2007-
11-09/index.html)

rev:Review a rdfs:Class;
        rdfs:comment "A review of an artistic work";
        rdfs:isDefinedBy <http://www.purl.org/stuff/rev>;
        rdfs:label "Review";
        vs:moreinfo "core term";
        vs:term_status "stable".

rev:hasReview a rdf:Property;
        rdfs:comment "Used to associate a work of art with a a review";
        rdfs:domain rdfs:Resource;
        rdfs:isDefinedBy <http://www.purl.org/stuff/rev>;
        rdfs:label "hasReview";
        rdfs:range rev:Review;
        vs:moreinfo "core term";
        vs:term_status "stable".

rev:rating a rdf:Property;
        rdfs:comment "A numeric value";
        rdfs:isDefinedBy <http://www.purl.org/stuff/rev>;
        rdfs:label "rating";
        vs:moreinfo "core term";
        vs:term_status "stable".

rev:reviewer a rdf:Property;
        rdfs:comment "The person that has written the review";
        rdfs:domain <http://www.purl.org/stuff/rev#Review>;
```

```
        rdfs:isDefinedBy <http://www.purl.org/stuff/rev>;
        rdfs:label "reviewer";
        rdfs:range foaf:Person;
        vs:moreinfo "core term";
        vs:term_status "stable".

rev:text a rdf:Property;
        rdfs:comment "The text of the review";
        rdfs:isDefinedBy <http://www.purl.org/stuff/rev>;
        rdfs:label "text";
        vs:moreinfo "core term";
        vs:term_status "stable".

rev:type a rdf:Property;
        rdfs:comment "The type of media of a work under review";
        rdfs:isDefinedBy <http://www.purl.org/stuff/rev>;
        rdfs:label "type";
        vs:moreinfo "core term";
        vs:term_status "stable".

################################################################################
# Import of the Time Ontology schema (from http://www.w3.org/TR/owl-time)

time:Instant
        a rdfs:Class.

time:inXSDDateTime
        rdfs:domain time:Instant;
      rdfs:range xsd:dateTime.
```

Annex C. An example with some log entries

```
@prefix dc:   <http://purl.org/dc/elements/1.1/>.
@prefix time: <http://www.w3.org/2006/time>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rev:  <http://www.purl.org/stuff/rev#>.
@prefix s:    <http://www.SOA4All.eu/log#>.
@prefix vs:   <http://www.w3.org/2003/06/sw-vocab-status/ns#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix :     <http://www.SOA4All.eu/log/entry#>.

############
:logEntry001
        a s:LogEntry;
        s:hasDateTime :logEntry001_Instant;
        s:fromIP "127.0.0.1";
        s:hasPersistentSessionID "qwerty12345";
        s:hasAction :logEntry001_Action.

:logEntry001_Instant
        a time:Instant;
        time:inXSDDateTime "2009-06-30T10:30:00+01:00".

:logEntry001_Action
        a s:LoginAction;
        s:loggedInUser :user001.

:user001
        a foaf:Person;
        foaf:name "John Doe".

############
:logEntry002
        a s:LogEntry;
        s:hasDateTime :logEntry002_Instant;
        s:fromIP "127.0.0.1";
        s:hasPersistentSessionID "qwerty12345";
        s:hasAction :logEntry002_Action.

:logEntry002_Instant
        a time:Instant;
        time:inXSDDateTime "2009-06-30T15:48:00+01:00".

:logEntry002_Action
        a s:LogoutAction.

:user001              # This may be avoided if already present
        a foaf:Person;
        foaf:name "John Doe".

############
:logEntry003
        a s:LogEntry;
        s:hasDateTime :logEntry003_Instant;
        s:fromIP "127.0.0.1";
```

```
        s:hasPersistentSessionID "qwerty12345";
        s:hasAction :logEntry003_Action.

:logEntry003_Instant
        a time:Instant;
        time:inXSDDateTime "2009-06-30T10:32:00+01:00".

:logEntry003_Action
        a s:ItemSelection;
        s:selectedItem <http://SOA4All.eu/service/DummyService001>.

############
:logEntry004
        a s:LogEntry;
        s:hasDateTime :logEntry004_Instant;
        s:fromIP "127.0.0.1";
        s:hasPersistentSessionID "qwerty12345";
        s:hasAction :logEntry004_Action.

:logEntry004_Instant
        a time:Instant;
        time:inXSDDateTime "2009-06-30T10:33:00+01:00".

:logEntry004_Action
        a s:ItemTagging;
        s:taggedItem <http://SOA4All.eu/service/DummyService001>;
        s:tag "Free SMS".

############
:logEntry005
        a s:LogEntry;
        s:hasDateTime :logEntry005_Instant;
        s:fromIP "127.0.0.1";
        s:hasPersistentSessionID "qwerty12345";
        s:hasAction :logEntry005_Action.

:logEntry005_Instant
        a time:Instant;
        time:inXSDDateTime "2009-06-30T10:34:12+01:00".

<http://SOA4All.eu/service/DummyService001>
        rev:hasReview logEntry005_Action.

:logEntry005_Action
        a s:ItemReview
        s:hasReview :rev001

### (This would be in the Feedback Framework)
<http://SOA4All.eu/service/DummyService001>
        rev:hasReview :rev001.

:rev001
        a rev:Review
        rev:minRating 0; # min and max ratings are fixed in SOA4All. We can
avoid these 2 lines from the log files. Anywway, they could be useful in
case we wish to publish the triples somewhere.
        rev:maxRating 5;
        rev:rating 4;
```

```
rev:text "Nice useful Service";
```