

Project Number: **215219**
 Project Acronym: **SOA4All**
 Project Title: **Service Oriented Architectures for All**
 Instrument: **Integrated Project**
 Thematic Priority: **Information and Communication Technologies**

D8.4. Web 21c Prototype v1

Activity N:	Activity 3 – Use Case Activities	
Work Package:	WP8 – Web21C BT Infrastructure	
Due Date:	M18	
Submission Date:	28/08/2009	
Start Date of Project:	01/03/2008	
Duration of Project:	36 Months	
Organisation Responsible of Deliverable:	BT	
Revision:	1.0	
Author(s):	Sandra Stinčić BT John Davies BT Marc Richardson BT Guillermo Álvaro iSOCO Freddy Lecue UNIMAN Nikolay Mehandjiev UNIMAN Maria Maleshkova OU	
Reviewers:	Sven Abels TIE Jacek Kopecky STI	

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	17.07.2009	TOC and initial structure outlined	Sandra Stinčić (BT)
0.2	29.07.2009	Description of services, description of customisations in scenario	Guillermo Álvaro (iSOCO)
0.3	30.07.2009	Section 2	Sandra Stinčić (BT)
0.4	31.07.2009	New Content in Section 4.2.2	Freddy Lecue (UNIMAN)
0.5	04.08.2009	Section 3	Nikolay Mehandjiev (UNIMAN), Sandra Stinčić (BT)
0.51	07.08.2009	Section 4.3	Sandra Stinčić (BT), Nikolay Mehandjiev (UNIMAN)
0.52	10.08.2009	Added details on 4.2.2 Customisation for the consumption	Maria Maleshkova (OU)
0.53	11.08.2009	Added content to Section 5, integrated Section 4.3 into Section 6	Sandra Stinčić (BT)
0.54	19.08.2009	Section 5.3	Guillermo Álvaro (iSOCO)
0.55	20.08.2009	Update to Figure 1, Glossary of Acronyms, Section 5.3.2	Sandra Stinčić (BT)
0.6	20.08.2009	Update to Section 4.1	Maria Maleshkova (OU), Sandra Stinčić (BT)
0.7	21.08.2009	Updates following comments received from Sven Abels (TIE)	Sandra Stinčić (BT)
0.71	25.08.2009	Updates following comments received from Jacek Kopecky (STI)	Sandra Stinčić (BT)
0.72	26.08.2009	Update to section 4.1.2.2	Freddy Lecue (UNIMAN)
0.73	26.08.2009	Updates following comments received from Sven Abels (TIE) and Jacek Kopecky (STI)	Guillermo Álvaro (iSOCO)
0.74	27.08.2009	Updates to Section 3	Nikolay Mehandjiev (UNIMAN)
0.75	27.08.2009	Updates to Section 4.1.2	Maria Maleshkova (OU)

0.76	27.08.2009	Updates following comments received from Alistair Duke (BT)	Sandra Stinčić (BT)
1.0	28.08.2009	Final version following the comments received from John Davies (BT)	Sandra Stinčić (BT)

Table of Contents

EXECUTIVE SUMMARY	6
1. INTRODUCTION	7
1.1 PURPOSE AND SCOPE	7
1.2 STRUCTURE OF THE DOCUMENT	7
2. SCENARIO OVERVIEW	8
3. EVALUATION WORKSHOP CONCLUSIONS/RESULTS	10
4. ARCHITECTURE OF THE PROTOTYPE	13
4.1 IMPLEMENTATION/CUSTOMISATION DONE FOR THIS DELIVERABLE	13
4.1.1 <i>List of Services used</i>	14
4.1.2 <i>Implementation Customisation Done for the Consumption of the Composed Service</i>	17
5. PROTOTYPE DOCUMENTATION	23
5.1 INSTALLATION AND USAGE	23
5.1.1 <i>For Users</i>	23
5.1.2 <i>For Administrators</i>	23
5.2 FRONT-END GUI	23
5.3 HOW TO USE THE PROTOTYPE	23
5.3.1 <i>Using the Scenario 1</i>	23
5.3.2 <i>Adding new Service(s)</i>	27
6. CONCLUSIONS	28

List of Figures

Figure 1: Scenario 1 Example	9
Figure 2: SOA4All Overall Architecture	13
Figure 3: Identifying service properties	18
Figure 4: Saving of the semantic RESTful services	19
Figure 5: Data Flow Model of the Scenario	20
Figure 6: “BT Scenario 1” shortcut located in the left-hand panel	24
Figure 7: Selection for Madrid example	24
Figure 8: Execution results for Madrid example	25
Figure 9: Selection for London example	26
Figure 10: Execution results for London example	27

List of Tables

Table 1 Attitudes of participants towards opening service composition to users	11
Table 2 Services used in Scenario 1	14

Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
BT	British Telecom
D	Deliverable
EC	European Commission
GUI	Graphical User Interface
hRESTS	HTML format for describing RESTful Services
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LPML	Lightweight Process Modelling Language
RDF	Resource Description Framework
REST	REpresentational State Transfer
QoS	Quality of Service
S1	WP8 Scenario 1
S2	WP8 Scenario 2
SDK	Software Development Kit
SMS	Short Message Service
SOA4All	Service Oriented Architecture for All
UDSD	User Driven Service Development
UI	User Interface
UniMan	University of Manchester
URL	Uniform Resource Locator
VoIP	Voice over IP (Internet Protocol)
WP	Work Package
XML	Extensible Markup Language

Executive summary

Work package 8, the BT Web21C case study, builds on the current BT Ribbit infrastructure, which provides a set of Web accessible Telco services, and leverages SOA4All research and technology to allow end-users to access, use and create services based on BT's 'capabilities' (such as VOIP, SMS etc.). Web 2.0 principles, Semantics and Context will be used to create a powerful but easy to use platform for discovering, consuming and combining BT's capabilities, and for incorporating third party services.

The case study has developed two scenarios as the basis for developing prototypes based on SOA4All technology. The prototypes will aim to provide the next generation of Ribbit, with version 1 of the prototype focusing on the novice users and ease of use of popular services on the web.

This deliverable outlines the development of version 1 of the prototype, with issues encountered. Design of the prototype follows the generic design of the SOA4All platform and has been explained in more detail in the D8.3 (Web 21C Futures Design), with the generic detail of the SOA4All Studio installation explained in the D2.4.2 (SOA4All Studio First Prototype).

1. Introduction

1.1 Purpose and Scope

The purpose of this document is to describe the first version of the WP 8 prototype (D8.4), built using technology developed by technical work packages.

This document

- briefly describes the WP8 Scenario 1,
- presents the results of the evaluation workshops to test user attitudes to user-driven service development,
- explains the architecture and implementation of the prototype,
- gives instructions on how to install and use the prototype

1.2 Structure of the document

The document is structured as follows:

- The following Section 2 provides a short summary of the requirements defined in D8.1 (Web 21C Futures Requirements) and design specification defined in D8.3 (Web 21C Futures Design).
- Section 3 describes the main conclusions of the evaluation workshops held by the University of Manchester and their impact on the WP8 prototypes.
- Section 4 explains how the WP8 prototype v1 fits into the generic SOA4All Architecture. In addition, customisations done to the services and consumption process are described
- Within Section 5 details about prototype installation, configuration and usage are given.

2. Scenario Overview

Scenario 1 (S1) from D8.3 (Web 21C Futures Design) describes a situation in which SOA4All technology is used for creating simple mash-ups of BT services and other popular services on the web. The aim is to make it easy for novice users to get access to the facilities of the Ribbit services and combine them with other services on the Web. SOA4All will be used to overcome some of the current problems that limit the uptake of the Ribbit services, primarily the technical knowledge required and familiarity with programming languages such as PHP or JavaScript. Scenario 2 (S2), also described in D8.3, details an “industrial strength” scenario.

Currently, usage of Ribbit services requires a detailed technical knowledge of both Web service languages and programming languages, in particular Adobe Flex, JavaScript and/or PHP. Although documentation and guides have been provided, it is not straightforward to implement a composite service even for an advanced user, let alone a casual user which S1 is aimed at.

The scenario involves building up Web Service composition to create a new web application incorporating Ribbit services. As the focus of S1 is on casual users building non-critical applications, the scenario will involve minimal security or management infrastructure. An example service composition that a user would create is outlined below. In this example, a composition is created that allows you to organise a meet up with a group of friends at the last minute (as shown on Figure 1).

- Get list of friends from social networking site – in this example Last.fm was used, as it has a simple and open API to retrieve connections between users, and one of the options for meeting up is related to concerts, which are also available in Last.fm.
- Find out which ones are in the area by using mapping sites such as FireEagle and Multimap
- Filter the meeting location list depending on the weather
- Find out travel information for the proposed meeting.
- Send out invites and directions using Ribbit SMS to those users that are within a set range, customised depending on the weather information.

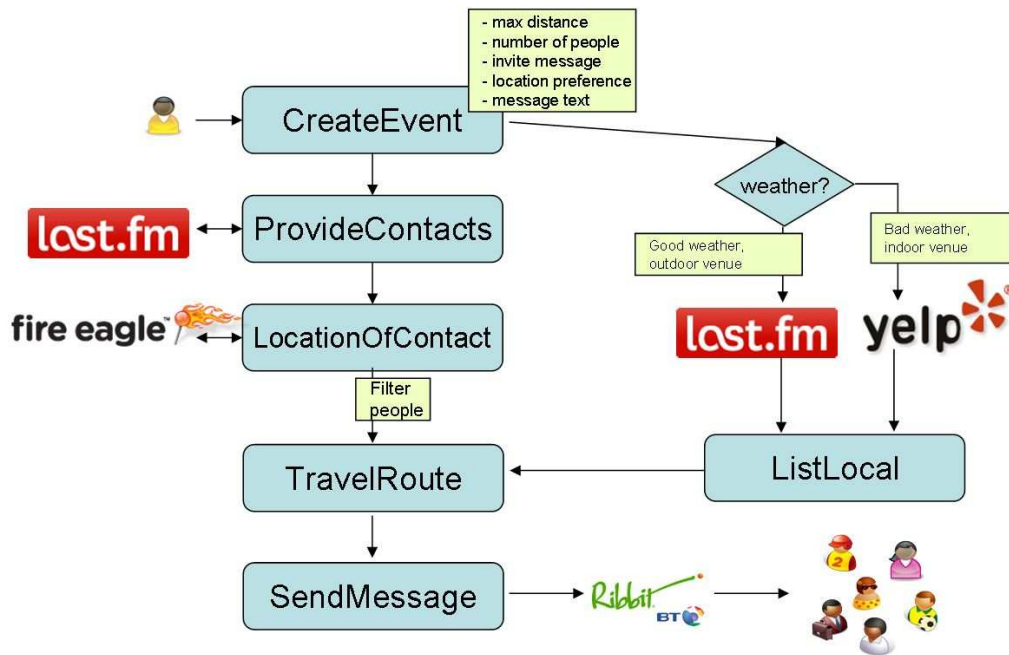


Figure 1: Scenario 1 Example

Users will undertake a similar sequence of steps each time they design a new application, although the specific services may differ.

Current limitations of the Scenario 1 include:

- Limited users' interaction in the composition of the services – e.g.:
 - Filtering of user's friends that have been listed as suitably located
 - Selection of the preferred venue (currently it is the closest/first listed one)
- To use the services/Scenario, one has to be a registered user of SOA4All Studio (registering with OpenID) and have its accounts on Last.fm and FireEagle manually mapped
- Temperature values that determine good or bad weather have currently been predefined: the decision value is 20 °C.

3. Evaluation workshop conclusions/results

A total of five workshops with 64 users were organised by the University of Manchester (UniMan), aiming to test the attitudes of the SOA4All target users to the ideas behind SOA4All in terms of user-led service development, and to identify main concerns as well as efficient ways to represent service development to end users. Three of the workshops were conducted with non-professional target users, the results of these are reported in D2.5.1. The other two workshops with a total of 29 users were based on professional users scenario as developed in WP8, and the results are reported below. Both workshops were organised jointly by BT and UniMan, the first was held at BT premises on the 12th May 2009, the second was held during the Semantic Week Conference on the 23rd June 2009¹. During each workshop, participants were engaged in a structured programme of activities, interleaving presentations, notational studies, focus group discussions and questionnaire completion.

The three main themes of discussion were (a) the business models enabled by the idea of opening up service development (including discovery, annotation and composition) to third parties and non-professionals, (b) risks and benefits from such user-driven service development (UDSD) for the target users and for their organisations and (c) discussing alternative representations and mechanisms to support UDSD. Each of the three themes is covered in a separate section, with business models covered in more details in the deliverable 10.3.1 (Stakeholders Consultation), Mechanisms for supporting users explained in the deliverable 2.5.1 (Formative Evaluation and User-Centred Design) and the benefits and risks of user-driven service development summarised below.

To estimate the likelihood of uptake by individual users of UDSD, the groups also reviewed the risks and the benefits which the target users associate with the idea. The major highlighted benefits were as follows:

- (a) The potential for saving time and reusing development effort was often regarded as the main factor motivating service development.
- (b) The opportunity to empower the creativity of third parties was also felt to be one of the main benefits. Combined with innovative charging models, this can be used for the commercial benefit of the service platform provider, which may offer value-added services such as validation and quality assurance, or charging for underlying resources such as communication bandwidth.
- (c) The third main benefit was the possibility to exploit a larger number of niche profit generation opportunities compared to conventional service provision models.

Perceptions of risks were focused on the following issues:

- (a) Privacy concerns, including the way personal data is handled and potentially passed to third party services under unknown jurisdiction and data protection rules. Transparency of data use was felt to be the key issue here.
- (b) Unexpected multiplier effect from community-style interactions. For example, a friend can provide your mobile number to an SMS service. More worryingly, this also exists for interactions without passing of data: the online “friend of a friend” style interactions may quickly create interaction patterns that may be considered security risk by authorities.
- (c) Preventing misuse in both security and financial terms was also considered a conventional risk that is multiplied with the widening of participation in the service

¹ <http://www.semanticweek.eu>

web. Trust may be helped by eBay-style feedback mechanisms, yet this is more likely to work in social setting, since it is easy to orchestrate positive feedback by unknown and possibly mal-intending people.

(d) Brand protection was seen as a key to organisational acceptance of the ideas of user-driven service development. In the context of Ribbit it was considered necessary to implement stringent validation practices, ensuring any third-party compositions provide the levels of service quality associated with the BT brand, especially since BT (via Ribbit) is providing core services in these compositions.

The attitudes of workshop participants (total of 29 participants on two separate workshops) towards the expected benefits and anticipated risks from opening up service composition to users were tested using the following six questions. The answers were provided on a Likert scale from (-2: Disagree) to (2: Agree), with (0) being the neutral answer.

Table 1 Attitudes of participants towards opening service composition to users

Service composition by users		Mean answer [-2..2]	StDev
Benefits	... is useful	1.44	0.76
	...brings about a more efficient way of conducting on-line activities	1.00	0.87
	...is easy to achieve	-0.11	1.22
Risks	... is unfeasible	-0.82	1.05
	... is error-prone	0.16	1.05
	... can be used to break organisational rules and policies	0.51	0.95

Their answers demonstrate a positive balance between benefits and risks, which is considered a necessary pre-requisite for uptake of such user-driven service composition. Evidently, service composition by users was considered to be potentially very useful, bringing about a more efficient way of conducting on-line activities. At the same time, it was not regarded as easy to achieve, yet its achievement was considered feasible. It was considered somehow error-prone with a non-zero risk to using it for breaking organisational rules and policies.

The aim of the workshops was to test the ideas underpinning SOA4All in terms of involving target groups of users in the overall development of software services, and especially on establishing the optimal mechanisms for supporting such activities. The conclusions and recommendations to WP8 are as follows:

- (a) Consider providing a two-tier interface, where experienced users are provided with detailed and powerful control-flow mechanisms for controlling service compositions, whilst novice users are provided with simple templates and asked to select pre-discovered services for each aspect of the template.
- (b) Provide examples of services composed by novice users, and a sandbox environment for other novice users to test and modify these examples.
- (c) Support the community dimension of the user development activities, using it

for sharing community-specific solutions and templates, for feedback on actual services used, and for general support and awareness.

(d) Provide facilities for template and service evolution, using a combination of (a) and (c) approaches above.

4. Architecture of the Prototype

The Architecture of the prototype is based on the overall architecture defined in WP1. A detailed description of this can be found in deliverable 1.4.1. The full architecture for SOA4All defines all components and their interactions (see Figure 2 below). Version 1 of the WP8 prototypes is focussed on a subset of the full functionality outlined in WP1. This is due to the specific requirements of the WP8 case study and taking into account the availability of some of the components developed in the project at the time the prototype was developed.

WP8 Prototype version 2 will more of the technology components from the technical work packages, especially in relation to Context and Web 2.0 principles, which will also be reflected in the complexity of the Scenario 2 (due M30). Work done in (amongst others) the WP5 (Service Location) and WP6 (Service Construction) related to Context will be used in the form of Service Rankings and Recommendations, customised Help during the design time selections, and more specifically to the scenario, usage of User Profile and Location to contextualise the SMS adverts generated in the Scenario 2. One of the important extensions in S2 will be usage of Monitoring, Quality of Service (QoS), Fault Handling and Error Reporting, which will be developed in the WP2 (SOA4All Studio). In addition, integration with various Web 2.0 technologies will be required, such as Importing and Mark-up of third party Web Services, which will be developed in WP2, WP3 (Service Annotation and Reasoning) and WP6.

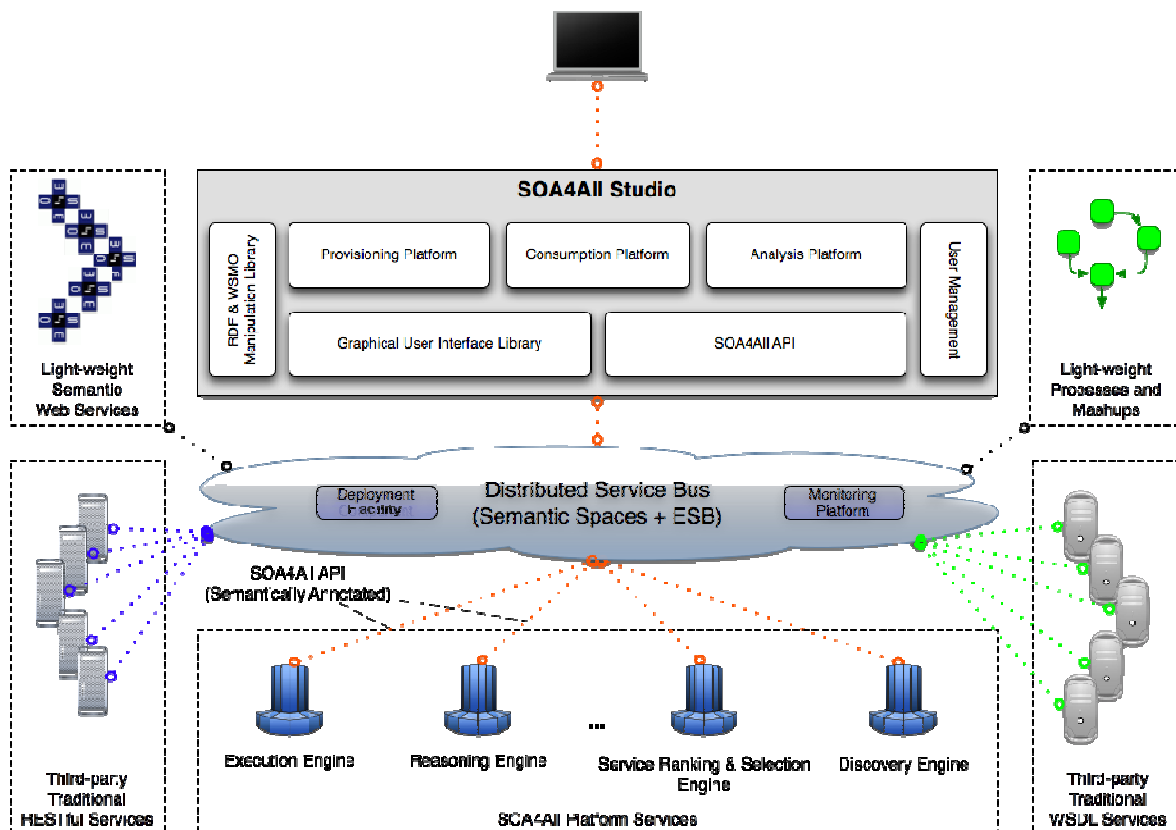


Figure 2: SOA4All Overall Architecture

4.1 Implementation/Customisation done for this deliverable

As at this stage of the project, all components required for the prototype are not mature

enough and therefore certain customisations need to be done in order to achieve Scenario 1. Details are given in the following sections.

4.1.1 List of Services used

For this scenario, several existing RESTful services, which express their functionalities via APIs were used. A summarised list of the services used is contained in Table 2 below, while the visual relation between the services is shown in Figure 1.

Table 2 Services used in Scenario 1

Service	Description
Last.fm	Provides a list of friends Provides concerts search
FireEagle	Provides user's location
Multimap	Provides driving or walking directions between a set of locations
Weather Bug	Provides live weather data
Yelp	Provides local search and retrieves a list of businesses
Ribbit	Provides send SMS service

The following sections briefly cover the most important characteristics of the services used.

4.1.1.1 Last.fm

Last.fm is a music community website focused on music recommendations, but at the same time enhanced by social networking features and extra music-related information such as concert listings, etc.

We will leverage their functionalities exposed via an API² in two different ways: By exploiting their social network and by retrieving events based on a given location.

The RESTful API requires an API key for authentication that third parties such as the SOA4All Studio have to apply for via their website. The following methods that we invoke just require the key to be sent as a parameter in the URL, and do not require extra authentication such as other more secure methods, which imply additional steps such as signing tokens:

→ user.getFriends

It retrieves the list of a user's friends in Last.fm. A "user" parameter is required, and the response is in XML.

Method description: <http://www.lastfm.es/api/show?service=263>

This method is important for the use case, for it implies that a user can leverage its social network built outside the SOA4All Studio.

→ geo.getEvents

It gets a list of events and venues for a given location. We will be using the optional

² <http://www.lastfm.es/api>

latitude (“lat”) and longitude (“long”) parameters to find these venues, returned in XML.

Method description: <http://www.lastfm.es/api/show?service=270>

4.1.1.2 Fire Eagle

Fire Eagle is a location-based service owned by Yahoo!, which is able to store a user’s location.

Users can authorize third-party applications to access their location thanks to their API³, which supports the OAuth⁴ protocol. This implies that before the SOA4All Studio access a given user’s location, it is necessary that he has authorized the SOA4All Studio into his Fire Eagle account, providing his Fire Eagle password only to Fire Eagle, but never to the Studio - which only gets an access token and a secret.

The token and secret pair will be used to get the user’s location later on, via the following method:

→ user

An external application can use this method to query Fire Eagle for the location of a user. The response can be formatted in either XML or JSON, depending on the choice. The relevant OAuth parameters (consumer key, token, nonce, timestamp, signature method, version, and signature) have to be passed in the URL in order to make Fire Eagle access the data of that particular user.

Method description: <http://fireeagle.yahoo.net/developer/documentation/querying>

Thanks to this method, users in the SOA4All Studio will be able to have their location specified in it just by linking their account to their Fire Eagle user.

4.1.1.3 Multimap

The Multimap web services include geocoding and routing capabilities, which can be integrated into third-party applications via their Open API⁵.

Multimap provides an API key that has to be included in the request URL in order to validate it against the service.

In our case, we are mostly interested into a particular operation (calculate the distance between two points) which is included into a method (routing) with a larger scope:

→ Routing

The Routing method allows generating driving or walking directions between a set of locations. Additionally, the XML response contains generic information about the route, such as the total distance, which is the response parameter we are interested in.

Method description: http://www.multimap.com/openapidocs/1.2/web_service/ws_routing.htm

4.1.1.4 WeatherBug

WeatherBug services provide live weather data to many applications via the WeatherBug

³ <http://fireeagle.yahoo.net/developer>

⁴ <http://oauth.net/>

⁵ <http://www.multimap.com/openapidocs/1.2/>

API⁶.

The API has several methods to retrieve weather information, with even forecasts and alerts, and additionally methods for retrieving locations and weather stations.

Regarding authentication, WeatherBug also issues an alphanumeric license key that needs to be included twice in the request URL.

The method we are interested in for this scenario is the following one:

→ [getLiveWeather](#)

This method retrieves the live weather based on different input parameters. We will make use of the Latitude and Longitude generic parameters in order to obtain weather data in XML for the specified coordinates, from where we can retrieve the concrete data about the temperature.

Method description: <http://weather.weatherbug.com/corporate/products/API/help.aspx>

4.1.1.5 Yelp

Yelp offers local search and user review services with social network features. These functionalities are offered for third-party developers through their API⁷.

Yelp provides a key for developers upon registration to their site, and this key has to be used in the URL in order to query any method.

For the use case scenario being depicted, this service offers an easy way to find alternative locations (when the weather is bad) to the concert venues.

→ [business review search](#)

This method can be used to retrieve different types of businesses, resulting in a JSON-formatted response with information about the places and the reviews made on them.

Method description: http://www.yelp.com/developers/documentation/search_api

In addition to stating the latitude and longitude parameters, in the request we will also specify the term we want to use for querying the Yelp directory, which in this case will be “bars”.

4.1.1.6 Ribbit

Ribbit offers a platform that allows developers of third party applications to interact with telephone networks, through different voice protocols and via text messaging.

These functionalities are exposed in an API⁸, which is currently moving from the SDK-downloadable type of API to a fully REST one, which involves the OAuth protocol for security.

Nonetheless, for the purposes of this demo, a single utility service has been abstracted in order to provide a direct way of integrating send-SMS functionality via REST invocations. The method is the following one:

→ [SendSMS](#)

This method can be used to send a text message to the specified number.

⁶ <http://weather.weatherbug.com/corporate/products/API/help.aspx>

⁷ <http://www.yelp.com/developers/>

⁸ <http://developer.ribbit.com/>

Method description: <http://ngwr.labs.bt.com/Ribbit/myapp/RibbitSMSService.php>

4.1.2 Implementation Customisation Done for the Consumption of the Composed Service

The current prototype that will be used for the demo of the first scenario of WP8 Use Case already highlights many important components of the project. However, given the early stage of some of the tools, platforms and components, and also considering the necessary changes that need to take place in order to accommodate the general SOA4All Studio to the requirements of the Use Case, we need to specify the customisations that take place at each stage.

This section covers the different stages of the scenario, and the customisations done in each of those, when applied.

4.1.2.1 Annotation of Services

The services used within this scenario are RESTful services with API descriptions in HTML pages. Before these services can be discovered and used in a service composition, they need to be supplemented with semantic annotation of their properties. This additional semantic information supports the automation of discovery, composition and consumption tasks, which otherwise have to be performed completely manually. In addition, as opposed to WSDL services, there is no widely accepted structured language for describing RESTful services. As a consequence, in order to use RESTful services, developers are obliged to manually locate, retrieve, read and interpret heterogeneous documentations of RESTful services in HTML, and subsequently develop custom tailored software that is able to invoke and manipulate them. A solution to these challenges is given by the first Provisioning Platform Prototype (D2.1.3), which as the name suggest, provides tools and functionalities for supporting the provisioning of services and in particular, semantic Web services. The creation of semantic RESTful services is enabled through the MicroWSMO editor, which is part of the Provisioning Platform Prototype. It enables both the creation of machine-readable RESTful service descriptions and the addition of semantic annotations, in order to better support discovering services, creating mashups, and invoking them.

Therefore, the first step of the scenario is to use the MicroWSMO editor and to create semantic descriptions of the RESTful services. Each of the service descriptions is annotated following these four main steps:

1. **Identifying service properties** -- Insertion of hRESTS microformat tags in the HTML service descriptions in order to mark service properties (operations, address, HTTP method, input, output and labels).
2. **Identifying suitable domain ontologies** -- Integrated ontology search for linking semantic information to service properties.
3. **Semantic annotation of service properties** -- Insertion of MicroWSMO model reference tags, pointing to the associated semantic meaning of the service properties.
4. **Saving of the semantic RESTful services** -- Saving of semantically annotated HTML RESTful service description and automatic extraction of RDF MicroWSMO service descriptions based on the annotated HTML into the SOA4All repository.

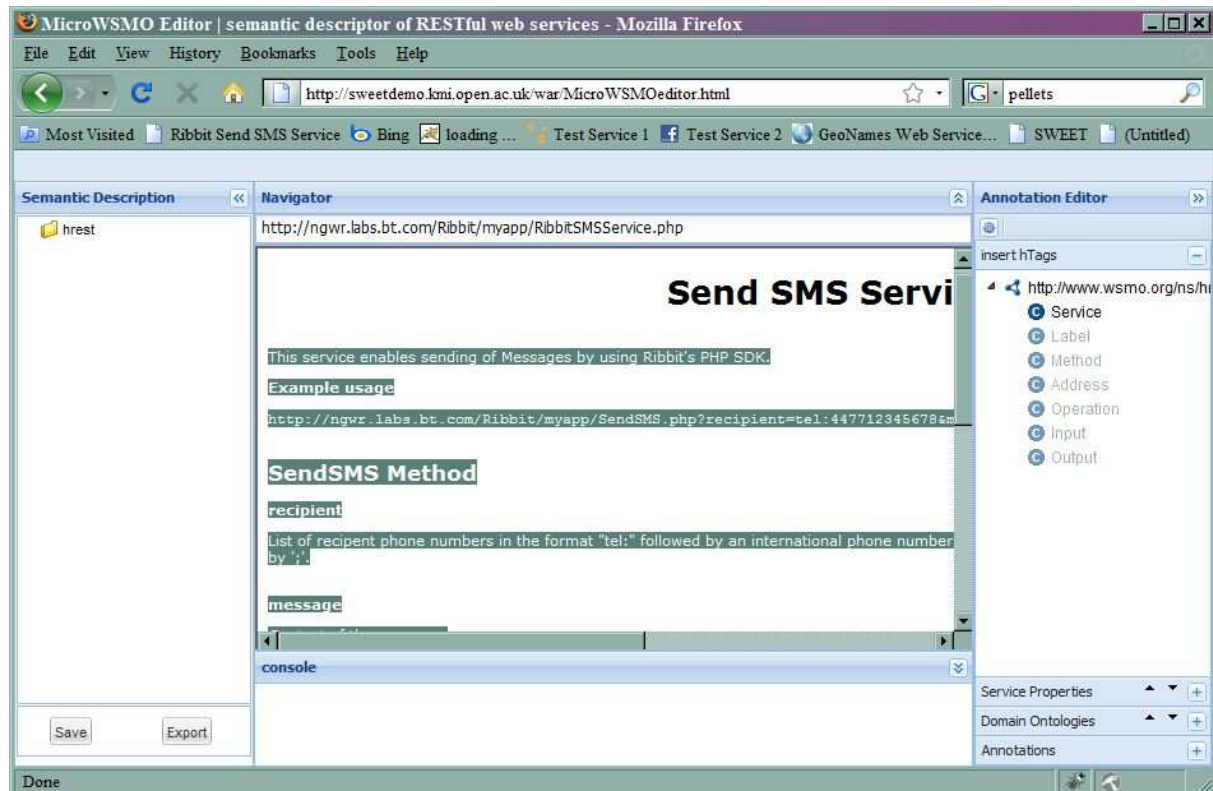


Figure 3: Identifying service properties

Step 1 is completely supported through the functionalities of the MicroWSMO editor and there were no customizations necessary (as shown in Figure 3).

The identifying of suitable domain ontologies was a bit more challenging and even though the MicroWSMO editor provides ontology search functions, sometimes the results were not suitable for the particular service in the scope of this use case. Therefore, in order to address this problem and to avoid the necessity of doing matching between ontologies, when services annotated with different ontologies are to be used in the same composition, we reused some widely used ontologies such as:

- <http://www.w3.org/2003/01/geo/> for latitude and longitude
- <http://www.service-finder.eu/ontologies/ServiceOntology> for service type classification
- <http://xmlns.com/foaf/0.1/> for person profiles
- <http://www.eyrie.org/~zednenem/2002/wail/> for geographical location
- <http://purl.org/NET/c4dm/event.owl> for events details
- <http://www.kanzaki.com/ns/music> for music events information

and engineered an additional one for the missing properties. With this small customization, the semantic annotation of RESTful service properties was completed without further difficulties, by simply using the MicroWSMO editor (more detailed description of the annotation of service properties can be found in the D2.1.3, Service Provisioning Platform First Prototype Documentation).

Finally, when the user is finished with annotating the HTML RESTful service description with hRESTS and MicroWSMO tags, the resulting annotated service can be saved or exported by clicking on the “Save” or “Export” buttons. Figure 4 shows that when the user clicks on “Save”, a pop-up window is opened, which allows him/her to choose a destination for saving the annotated HTML. The export provides a RDF transformation of the annotated HTML. The

result is the MicroWSMO description of the semantically described RESTful service. In the future implementations of the editor, SWS can be stored through the Storage Services in the Semantic Spaces, where they can be reused and discovered by other users.

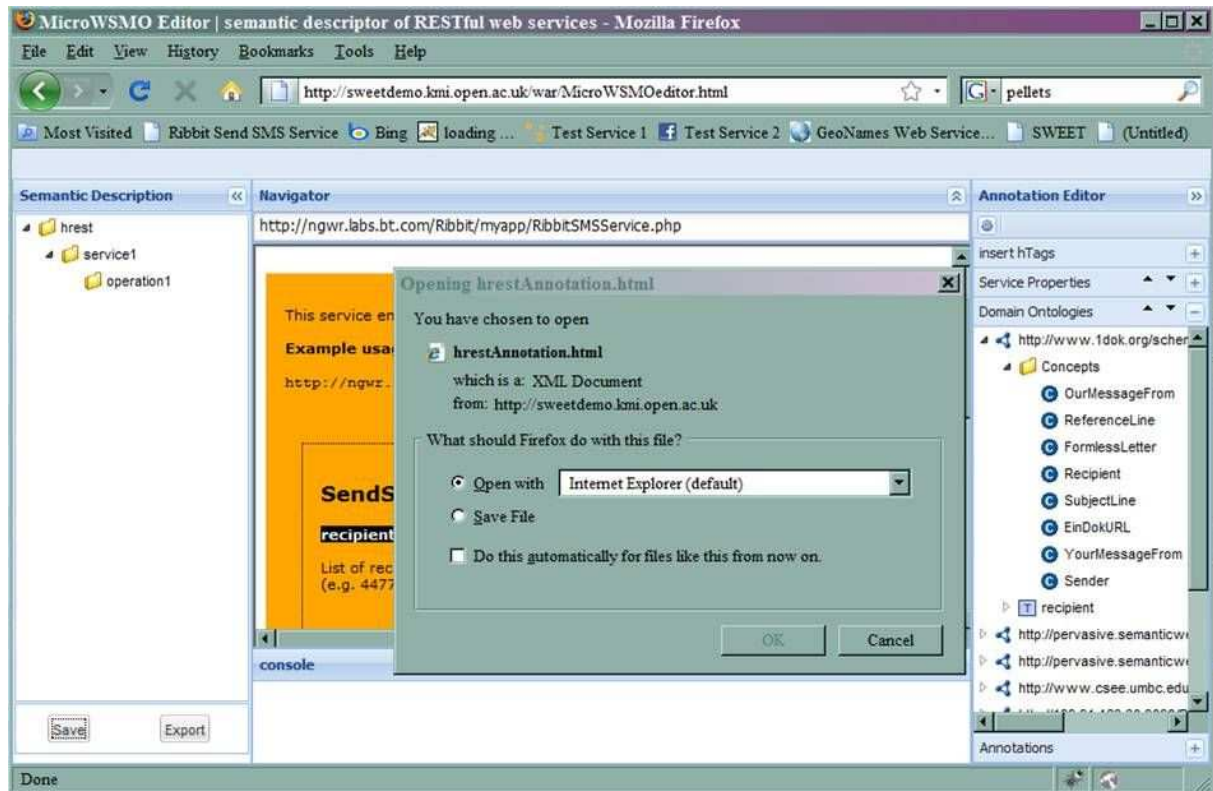


Figure 4: Saving of the semantic RESTful services

4.1.2.2 Composition of the process

The Process Editor tool is used to link the different (annotated) services into the desired process, also taking into account the restrictions that apply in the scenario. The latter process is mainly composed of mashing data (red - patterns filled bar - components in Figure 5) from one service to another (see Figure 5), for instance:

- Splitting data from the end user inputs e.g., “Location User” is reused in different points of the composition;
- Merging data from different services’ outputs e.g., the `getLocalVenue` and `getLocalBar` services provide some outputs that require to be merged for processing them in the next step (by some following services);
- Filtering data from the output of a service e.g., the `SendSMS` service will need only the first Location venue provided by `getLocalVenue` and `getLocalBar` services’ outputs.

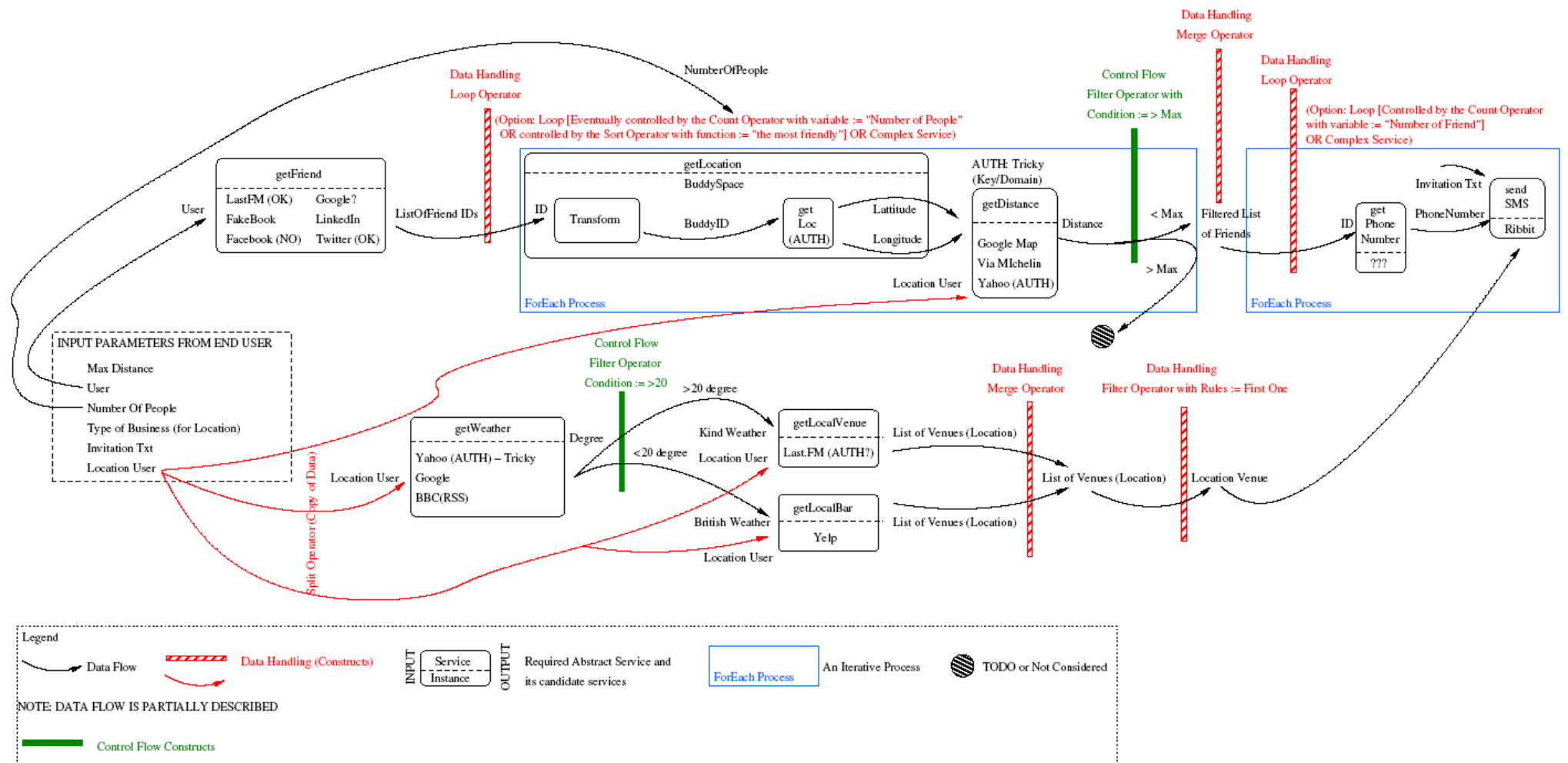


Figure 5: Data Flow Model of the Scenario

- Iterating processes on specific data controlled by some rules e.g., each element of the getFriend service needs to be processed by the getLocation services as inputs. The rule used is “Count Operator” with the “number of people” as variables.
- Counting the number of output provided by a service e.g., the getLocation service iterates on a given number of outputs of services, here a given number of friends.

In addition, the latter process is composed of one specific control flow construct (green - fully filled bar - components in Figure 5) supported by the LPML (Light-weight Process Modelling Language introduced in Deliverable D6.3.2 of SOA4All):

- Conditional branching on services depending on the values of output data of services e.g., the getWeather service moves to getLocaVenue or getLocalBar depending on the data provided by getWeather service.

To this end, the scenario has been modelled and encoded using the LPML composition language provided by T6.3. In more details, the whole composition has been specified using:

- A subset of the “Data Flow” operators (enabling to model composition as a Mash-up of data) provided by the defined language:
 - Split Operator: This operator receives an input and splits it into two or more identical outputs. This operator can be used when the end-user wants to perform different operations on data from the same input.
 - Merge Operator: This operator takes an arbitrary number of inputs and produces an output, which is composed of the merge of its inputs.
 - Filter Operator: This operator can be used to include or exclude items from an output of a service. Therefore, some rules can be created on top of the LPML language to compare the output of services to values that the end-user specifies.
 - Loop Operator: This operator introduces the idea of sub-data processing. Any other operators could be inserted inside the Loop operator. An output of a service is provided to the Loop operator, the sub-data processing is run once for each item in the latter output.
 - Count Operator: This operator counts the number of items in the input and outputs that number.
- A subset of the “Control Flow” operators provided by the defined language:
 - If_Then_Else Operator: This operator enables to branches one service to another depending on some logical conditions.

Work regarding the composition language (LPML) and specially the tool is still in progress, so some parts of the composition will be shown, but the actual composed process will not be executable yet.

4.1.2.3 Consumption of the composed process

For the scenario to be completed, it is necessary to make the invocation of the process possible within the Consumption Platform. So far, the platform is more focused on the consumption of single services (reason being because there are not yet available service compositions to be consumed), but for the purposes of this demo we will enable a way of invoking the scenario composed service.

As the Consumption Platform already permits interacting with single services, creating the input forms based on the annotations previously assigned to them, the composition of the desired process within the platform is hard coded by linking the outputs of some of the services to the inputs of others at a conceptual level. Looping and branching when necessary

is done by coding “for” and “if-else” statements.

Another customisation needed so far, to make the whole scenario work, has to do with the lowering and lifting mechanisms that are needed to move from the conceptual layer to the execution layer underneath. At this stage, there is not a way for manually creating lifting and lowering schemas in an easy manner, and in addition, the integration with the proper lifting and lowering external services is still at an early stage. As a result, the necessary mappings for lifting and lowering in this scenario have been manually created in the code, ensuring that the consumption of those semantically enriched services can take place completely.

Regarding the authorization methods that arise when interacting with REST services that require so, we are interacting with two kinds of authorization:

- API key to be sent in the URL: In this case, we have previously asked for the keys and stored them conveniently in order to be able to interact with those services.
- OAuth: For this, we have previously retrieved the access token and secret for the users participating in the scenario.

Hence, one last customisation has to do with the users of the scenario. As stated, the OAuth access tokens are previously retrieved and associated to the test profile accounts within the SOA4All Studio. They are also linked manually to Last.fm usernames and users’ phone numbers so the scenario can take place. However, as there is already a User Profile within the SOA4All Studio (see D2.4.2), we expect to open the possibility of associating FireEagle and Last.fm users, telephone number, etc., with a SOA4All account from there, thus allowing any new user to register and test the scenario with his own data and social network characteristics.

5. Prototype Documentation

In this section installation and usage instructions for the prototype are given.

5.1 Installation and Usage

This section contains a short overview on how to install and use the current WP8 prototype. No software licences are required for running the prototype.

5.1.1 For Users

Only thing needed to use the WP8 prototype is a modern Web Browser such as Firefox or the Internet Explorer. No further installation is needed and it is not necessary to install any plugins. Based on this, users may simply invoke the SOA4All Studio application by calling the web address.

The D8.4 prototype is currently (28.08.2009) available for testing purposes at:

<http://coconut.tie.nl:8080/soa4all>

However, this will be moved to the official SOA4All website, which is currently in the process of the development.

5.1.2 For Administrators

Detailed information for administrators, including how to install and deploy a new instance of SOA4All Studio, is contained in the D2.4.2 (SOA4All Studio First Prototype).

Everything required for the WP8 Scenario 1 is included in the default SOA4All installation.

5.2 Front-end GUI

The user Interface for WP8 follows the standardised UI defined in WP2, as described in the deliverable D2.4.1 (SOA4All Studio First Demonstrator + Interface Specification).

Currently, there were no changes to the standard SOA4All Studio 'look and feel'. For the future development additional UI components might need to be developed (or standard ones customised) for some specialised tasks that are not provided by the standard SOA4ALL studio. All additional UIs will adhere to the standard 'look and feel' defined in D2.4.1.

5.3 How to use the prototype

5.3.1 Using the Scenario 1

The running version of the whole SOA4All Studio (currently available at <http://coconut.tie.nl:8080/soa4all>) includes a shortcut for consuming the composed process explained in this deliverable.

The shortcut is located in the bottom part of the "My Actions" area in the left-hand panel of the Consumption Platform (accessible by clicking "Consume" in the main dashboard). By selecting the "BT Scenario 1" link depicted in Figure 6, a "portlet" containing the service will be opened in the main portal area.

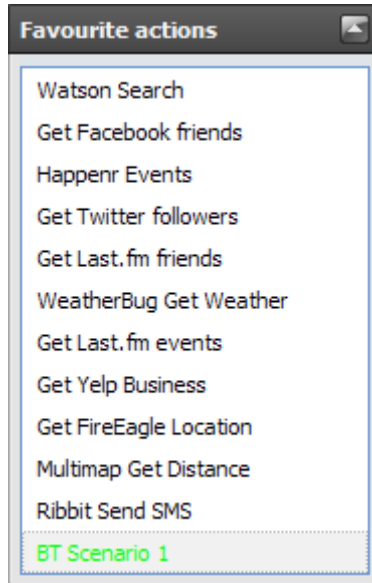


Figure 6: “BT Scenario 1” shortcut located in the left-hand panel

Once the service is displayed in a box, we recommend maximising it to the full portal size, using the square button in the upper right corner, in order to interact better with the service.

To do so, any user can introduce the desired coordinates (latitude and longitude). This information might be part of the context of the user in the future, but right now for this scenario, one can actively specify the place. In addition, for the scenario, the user does not have to be logged in into the SOA4All Studio, but just provide his Last.fm username, as we are taking advantage of its external social network built in other service.

In the example depicted in Figure 7, a user who is in Madrid introduces his Last.fm name and the text that he wants to include in the message.

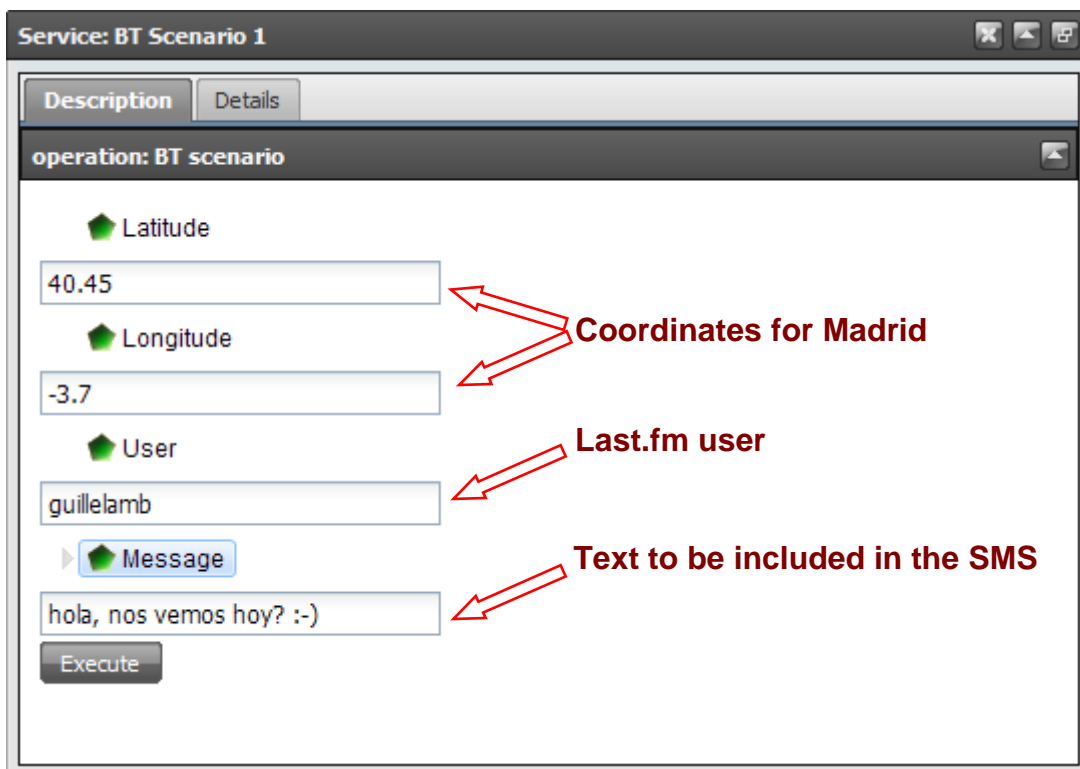


Figure 7: Selection for Madrid example

Once the “Execute” button is clicked, the composition is invoked, and eventually the results are displayed in another tab. Figure 8 depicts these results with the given parameters:

- The user has 6 friends in Last.fm.
- 4 of these Last.fm users are registered in the SOA4All studio.
- The weather in Madrid is good (temperature is over 20°C).
- Last.fm concerts are retrieved.
- The first of those venues is selected.
- Two of the friends are nearby. Another one is far away and there is no location data about the other.
- Hence, two SMS are sent.

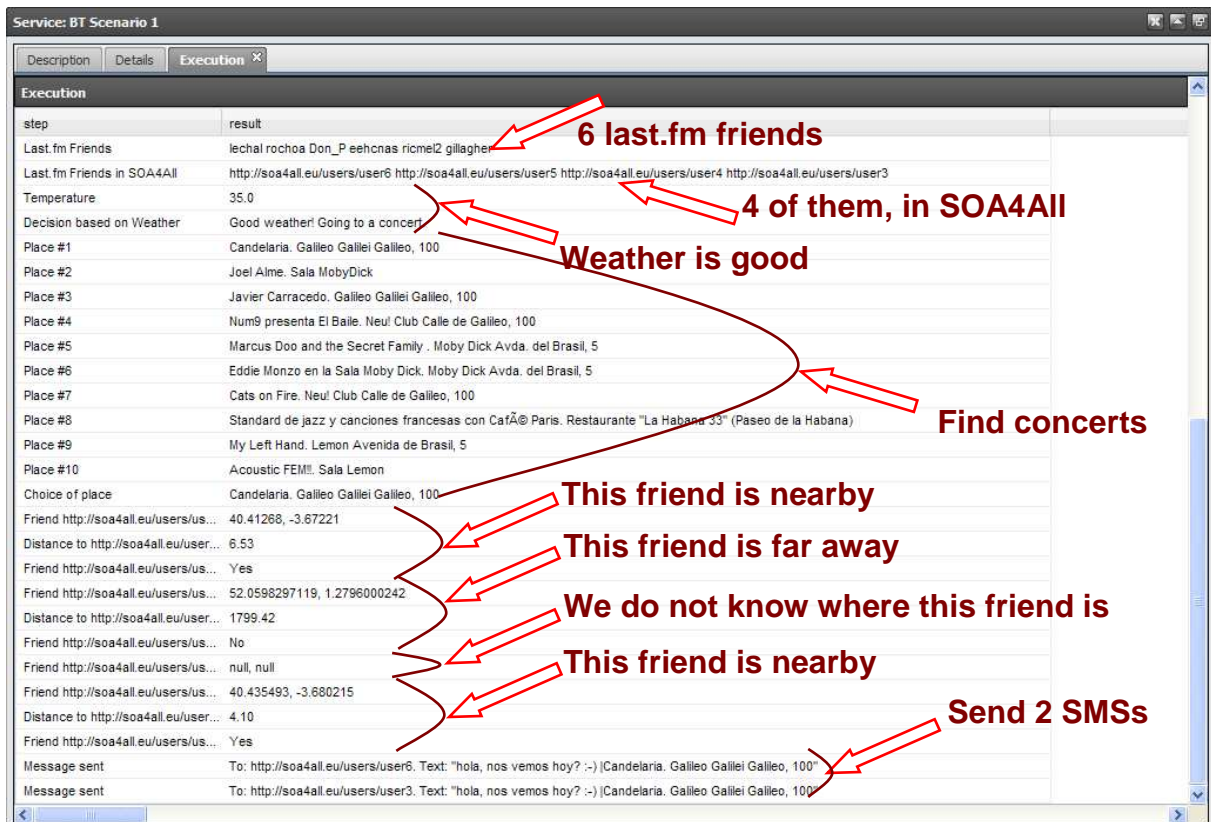
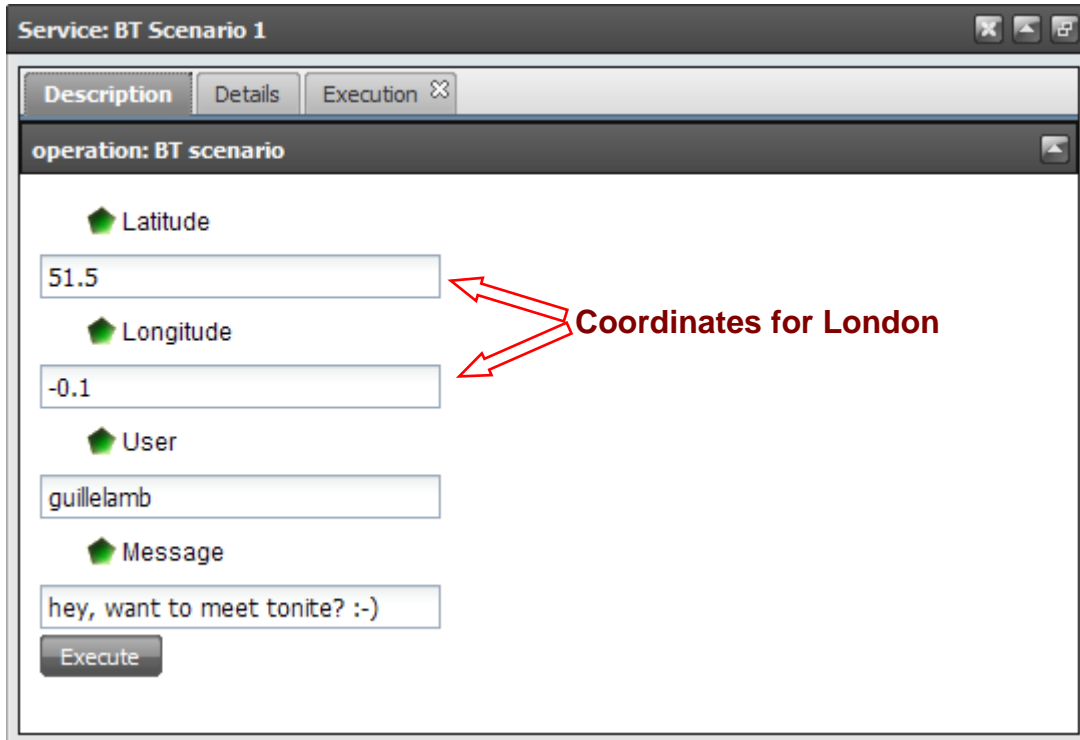


Figure 8: Execution results for Madrid example

In order to further illustrate the prototype with the other possible scenario with the alternate execution (when the weather is bad), we will suppose that the same user has travelled to London on the following day, and invokes the service again selecting new coordinates. Figure 9 shows the new selection of parameters.



Service: BT Scenario 1

Description Details Execution ✕

operation: BT scenario

Latitude
51.5

Longitude
-0.1

User
guillelamb

Message
hey, want to meet tonite? :-)

Execute

Coordinates for London

Figure 9: Selection for London example

By selecting different coordinates, the execution of the composed service will have a different result. In this case, the steps depicted in Figure 10 (where the differences with the other example are highlighted) are the following ones:

- The user still has 6 friends in Last.fm.
- Again, 4 of these Last.fm users are in the SOA4All studio.
- The weather in London is not good (temperature is below 20°C).
- Yelp list of bars is retrieved.
- The first of those bars is selected.
- Only one of the friends is nearby. Another two are far away and there is no location data about the other.
- Hence, only one SMS is sent.

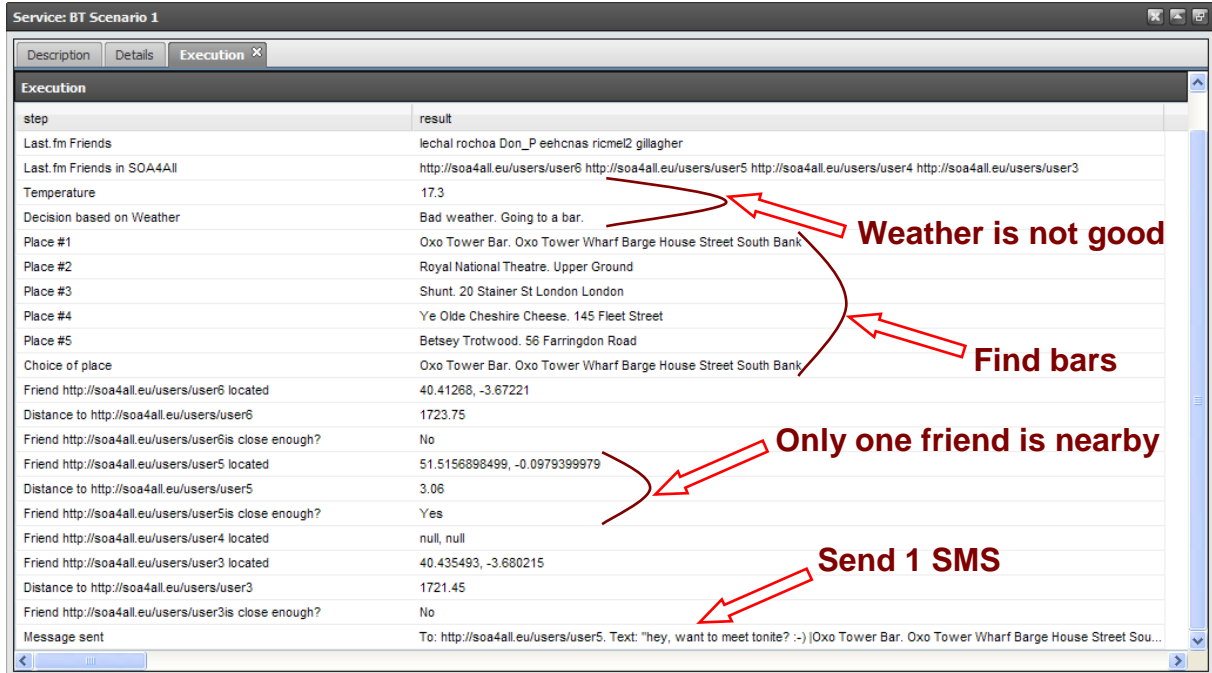


Figure 10: Execution results for London example

Of course, the two examples described here can be modified by selecting a different location, text, or a different Last.fm user. Trying some of the other usernames that appear in the examples is a good idea, as some of their connections have already their FireEagle accounts linked to the SOA4All Studio. In addition, we plan to open the integration of more users with the platform in the short future.

5.3.2 Adding new Service(s)

Currently, adding of a new service to be used within SOA4All Studio is not straightforward – that will change with the further evolution of the platform, and be shown in the second scenario (S2). To add new services to the scenario, one has to do the following:

- Annotate the service with semantic annotations (as described in the Section 4.1.2.1 of this document). More details about usage of the MicroWSMO Editor and service annotation can be found in the deliverable D2.1.3 (Service Provisioning Platform First Prototype).
- Link the annotated service into the desired process by using the Process Editor Tool (as described in the Section 4.1.2.2 of this document).
- Enabling invocation of the process within the Consumption Platform (as described in the Section 4.1.2.3 of this document).

6. Conclusions

This document has described the key results of the task 8.4. It presents the first version of the WP8 prototype, implementing Scenario 1 of the BT use case. Deliverable D8.3 (Web 21C Futures Design) has helped to accomplish this task as it guided the development and the implementation of the current prototype.

Scenario 1 has been successfully implemented using SOA4All Studio and can be invoked and run via any browser. Inevitably, at this stage of the project, not all technology components were available for use.

Based on the analysis provided in task 8.2 and the results of the prototype case studies, the next task for WP8 is to provide solid recommendations on how the transformation to a 'Future Telco' can be successfully achieved. The aim of this task will also be to engage other telecommunication providers to disseminate the results and recommendations from the task. This will be done through organisations such as the Telemangement Forum and relevant industry events and conferences.

For Scenario 2, as a business scenario, additional requirements such as the greater level of control over the execution of services, monitoring and fault handling have been identified. In S2, businesses will use SOA4All technology to design and compose more complex end user applications to resell or use as part of their business, incorporating BT white label Ribbit services, their own services & OSS and some BT OSS services. This will enable people to create a business incorporating BT services, without complex face-to-face contract negotiations and manual work to integrate services. At the same time, it will enable businesses to go from 'idea to product' in significantly reduced time.

A parallel activity will be prototype v1 usability evaluation, which will feed into D2.5.2. In addition, WP8 will provide fit-for-purpose evaluation of module features and technical WP results in terms of the WP8 scenarios.