Project Number: **215219**

Project Acronym: **SOA4All**

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

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D8.6 Web 21c Prototype v2

Activity N:	3
Work Package:	8
Due Date:	30/11/2010
Submission Date:	30/11/2010
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	BT
Revision:	1.0
Author(s):	Alistair Duke (BT), John Davies (BT), Carlos Pedrinaci (OU), Jacek Kopecký (OU), Guillermo Álvaro Rey (iSOCO)
Reviewers:	Sven Abels (TIE), Georgi Pavlov (SAP)

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	18.11.2010	First draft	Alistair Duke , John Davies, Carlos Pedrinaci, Jacek Kopecký, Guillermo Álvaro Rey
0.2	22.11.2010	Second draft for internal review	Alistair Duke
0.3	29.11.2010	First release version following internal review	Sven Abels (TIE), Georgi Pavlov (SAP)
1.0	30.11.2010	Final revision format	Julia Wells (ATOS)

Table of Contents

Table of Figures

Glossary of Acronyms

Acronym	Definition
API	Application Programme Interface
CeBP	Communication-enabled Business Process
D	Deliverable
EC	European Commission
EE	Execution Engine
GUI	Graphical User Interface
I/O	Input/Output
PE	Process Editor
ReST	Representational State Transfer
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SAWSDL	Semantic Annotations for WSDL
SDP	Service Delivery Platform
SOWER	SWEET is nOt a Wsdl EditoR
SMS	Short Messaging Service
SPARQL	SPARQL Protocol and RDF Query Language
SPICES	Semantic Platform for the Interaction and Consumption of Enriched Services
SWEET	Semantic Web sErvice Editing Tool
UML	Unified Modelling Language
WSDL	Web Services Description Language
WSMO	Web Services Modelling Ontology
XML	eXtensible Markup Language
XSLT	XML Stylesheet Transformation

Executive summary

This deliverable describes the 2nd prototype of the BT Case Study in SOA4All.

We introduce the Offers4All scenario as an example of a many-sided business model adhering to the Telco 2.0 approach aspired to by Telcos. The need for and use of SOA4All technology in the prototype are described.

Finally we look ahead to the remainder of the project where we will evaluate the technology against the requirements of BT Strategy's work on Communication-enabled Business Processes.

1. Introduction

This deliverable describes the second prototype of the BT Case Study. The aim of the case study is to show how SOA4All technology can enable telecommunications companies and their partners to generate value through building and offering innovative services to their customers in an agile way. SOA4All technology delivers the following key benefits:

1. It will allow such services to be developed and updated with greatly reduced software development effort;
2. It will promote open innovation by 3rd parties, increasing the exploitation of services by widening the number of people and organisations who are able to partake;
3. It will allow services to be operated in a scalable, efficient manner;
4. Finally, the above will contribute to reduced time-to-market and costs and a higher quality of service.

Closely related to this is the Telco 2.0 initiative as described in D8.5 (Telco 2.0 Recommendations). Here telecommunications companies (telcos) are looking to develop many-sided business models by allowing customers and partners to create value by building services that i) are based around telecoms infrastructure exposed via APIs; ii) are built using service creation environments and hosted on service delivery platforms and iii) exploit the relationship with and knowledge about the millions of customers of the telco.

Within BT we have aligned the SOA4All project with BT Strategy's work on Communication-enabled Business Processes (CeBP). CeBP is an embodiment of BT's effort to adopt the Telco 2.0 initiative. The central theme of CeBP is that supporting business processes is a higher value exercise than providing a pure communication channel i.e. if telcos can ensure that business processes result in a satisfactory outcome, then customers will be willing to pay more for the communication channel than they would otherwise. By allowing communication to be closely integrated into the business process and by using the knowledge that telcos hold about their customers, telcos can increase the efficiency of processes and reduce the friction that often results between service providers and their customers.

In our first prototype we focussed on using SOA4All technology to allow general users of the web to create telecommunications-based service mashups for their own use or for use in their community. In the second prototype described in this deliverable, the emphasis is very much on supporting businesses to create value generating services for their customers. It is in this kind of scenario that we see the greatest opportunities for exploitation of SOA4All technology within BT.

This deliverable is structured as follows. Section 2 provides a description of the Offers4All scenario we have adopted in the case study. This has a many-sided business model at its core and includes business processes enabled through communications. Section 3 describes the prototype we have developed which illustrates how SOA4All technology can be used to realise the Offers4All service. In Section 4 we outline the next steps including further planned use of SOA4All technology and how we will evaluate and exploit the work in BT.

2. Offers4All Scenario

In this section we introduce the Offers4All scenario, describe how this would be realised without SOA4All technology and the problems therein and then show the benefits of the application of SOA4All technology and how it addresses the problems identified.

We have developed the Offers4All scenario as an example of how BT and a partner (which could be a third party or a product line of BT) can collaborate to develop a business with communication enabled processes at its heart. SOA4All technology will greatly enhance such a business by allowing the partner to quickly build, reconfigure and deploy business processes that integrate BT and third party services.

Offers4All could be a service offering of a retail division of a telco such as BT Retail or a non-telco company. For the purposes of the description we will assume that Offers4All is provided by a company OfferNet who in the Telco 2.0 sense can be seen as an upstream customer of BT, that is they are using BT's infrastructure to generate value for end users (who can be described as downstream customers).

The Offers4All service allows companies e.g. retail organisations, entertainment providers, travel / hotel companies to advertise offers to subscribers of the service. These offers might be "last-minute" travel or entertainment deals or predefined campaign offers from retail organisations. The Offers4All service allows an offer provider to create a new offer by describing what the offer is and who and how many people it wants to target with the offer. An appropriate set of subscribers are then chosen and are made aware of the offer via a communication channel. Offer providers pay to use the service but subscribers do not. Possible charging models for offer providers include:

- Flat rate per offer
- Variable rate based on number of people contacted
- Variable rate based on number of people contacted and communication channel used
- Variable rate based on number of conversions to sales
- A combination of the above

When generating the offer, the offer provider describes the offer they wish to make including:

- A categorisation of the offer (using an Offers4All taxonomy)
- When (dates and times) the offer should be made
- Number of subscribers to make the offer to
- Nature of subscribers to make the offer to e.g. target age group, gender, location, interests, salary range
- Content associated with the offer e.g. text description, audio content, multimedia content
- Preferred communication channel
- Terms and conditions of the offer including instructions for taking up the offer e.g. voucher codes, sales channel, etc.
- An indication of whether the offer should be specific to a user (e.g. via a unique code) or whether it can be shared by subscribers i.e. allowing viral distribution.

Based on this information a set of subscribers are chosen to receive details of the offer. On joining the service subscribers will be invited to provide information on their interests and

willingness to receive certain kinds of offers. This will be incorporated with data already held by BT such as address, age, devices and contact details to form a profile. The profile is then used to match subscribers with appropriate offers and determine an appropriate communication channel for the offer message. A key element of value add for the service from BT is that the communication channel could be selected based upon what BT knows about the current status of the user e.g. they may have signed up to location based services allowing BT to determine their location and suitability for the offer on those terms. Alternatively, they may have recently used their home phone and would probably be contactable via this medium. The service is attractive to customers because they get to hear about offers that are relevant to them using the medium that they prefer but also because they do not need to share any personal details with offer providers. Since their details are held by one central trusted party i.e. BT, they have more control over the flow of information to them and can be sure that their personal data is not abused.

The data that BT holds about its customers is an important differentiator which allows them to retain competitive advantage in this kind of scenario vs. a new entrant who would just use basic communication channels. By communicating with users via the most appropriate device and increasing the likelihood of a successful communication occurring, more value can be attached to the service.

In addition to the central offers distribution service, additional services will be required to allow offer providers to track the take-up of offers and adjust the nature of existing offers and to allow subscribers to adjust their profiles and browse for existing offers. OfferNet will require monitoring services to assess quality of service, conversion rates, etc.

In considering how a service such as Offers4All could be built using existing technology, we note the following trends:

- Telcos are providing access to their services and infrastructure via publicly available APIs (as described in deliverable D8.5), an example of this being BT's Ribbit.
- 3rd parties i.e. non-telcos also expose their services via APIs e.g. Facebook, Amazon, Twitter, Google, etc. These are typically SOAP/WSDL or RESTful APIs but could also be abstractions of these based around programming languages such as Java or PHP.

In order to build the service, OfferNet must identify the existence of the various APIs that meet their needs. The amount of APIs and services exposed is considerable which is complicating the location of suitable services. Existing approaches for discovery tend to be rather ad hoc e.g. searching for a page describing the API. Once identified they need to understand how to interact with the API itself and also ensure they have an understanding of the supporting technology or programming language which of course differs widely amongst service/API providers. There are also various authentication approaches which must be adhered to. Typically, an organisation will have a product or solution designer who will design and describe the requirements of an application or service including the necessary components. It is then up to software developers to take the design and generate an implementation based upon it. The resulting software is then validated to ensure that the requirements are met. This is often an iterative process. In addition, subsequent changes to the design require that the process is carried out again.

Once the software development process is complete, the service must be deployed. OfferNet could choose to use their own infrastructure or they may make use of virtualised infrastructure and there are positives and negatives in both of these approaches. However, both of them have the drawback that the deployment, management and monitoring of the service itself (rather than the infrastructure) is left to OfferNet and they are faced with the need to deploy and run tools to allow them to carry out these functions. More support is provided by Service Delivery Platforms (SDPs) that are offered by telcos and others.

However, today's SDPs tend to be optimized for the delivery of a service in a given technological or network domain rather than focused on supporting multi-disciplinary services operating on diverse networks.

More generally, the dynamicity of the Web is such that there is a race for innovation, whereby companies are trying to release new innovative solutions with a significant anticipation of potential competitor activity. This dynamicity will also call in the future for highly adaptive technical solutions so that changes on used APIs or even their disappearance won't provoke failures of otherwise successful solutions and business models. Existing software development approaches for building such solutions would appear to be too slow, brittle and expensive.

2.1 SOA4All Benefits

We will now reconsider the Offers4All scenario with the use of SOA4All technology. The key aim is to show that by offering a platform based around the technology, BT can massively ease the complexity faced by partners such as OfferNet and its own market-facing units in developing and deploying services. A BT platform based on SOA4All technology would enable the creation of a community with an interest in creating communications related services. Such a platform would provide OfferNet with access to a service repository and discovery mechanism based on semantics, allowing them to identify services and APIs that meet their needs. Obviously, this is dependent upon the services being semantically described. The assumption here is that BT's own services such as those offered by Ribbit would be semantically described by people in BT with the appropriate skills. 3rd party i.e. non BT services and APIs would either be described by BT in the case where the service is seen as vital in supporting platform users' requirements or by the 3rd party service provider where they are keen to increase take-up of the service via the BT platform. In the future, as the number of services offered grows, having proper annotations will be a competitive advantage in itself allowing more people to retrieve them and make use of them.

The effort required is thus transferred from OfferNet to BT as the platform provider or to the 3rd party service provider. A key point though is that the effort for the description is required only once compared with the much larger effort required for each partner to develop software against the service's (non-semantic) API. As the use of these kind of services grow, more and more annotations will be available based either on provider annotations or simply on third parties like OfferNet who have gone through a similar process. Annotating them at this stage should require less effort arguably.

The BT platform based on SOA4All technology will also include a service creation environment providing non-software developers with the ability to create processes using an intuitive interface. The service descriptions allow the complexity in building processes from multiple services using diverse technologies and approaches to be diminished. This can greatly reduce the time and cost involved in developing and launching new applications. The product designer is able to generate the necessary processes themselves without the need to write down requirements for software developers to code against. The iterative cycle for the creation of processes is removed as are the communication issues involved in transferring requirements between people.

Once the processes have been developed they can then be deployed using a Service Delivery Platform including SOA4All's federated infrastructure. This will allow OfferNet to deploy in a manner appropriate for their needs (e.g. ensuring that latency which is an issue with real-time services can be reduced), to manage the deployment based on changing requirements and to monitor aspects of service delivery such as quality of service.

The end result is that the technology enables OfferNet to develop services in a more cost effective and timely manner and increases the likelihood of the many-sided business models

aspired to by telcos being successful.

3. Description of Prototype

In this section we describe the M33 prototype based on the Offers4All scenario. In developing the prototype we have deemed it necessary to act simultaneously as BT and OfferNet answering the questions: 'What does it take for BT to provide a platform allowing partners to create processes based on BT and 3rd party services?' and 'What does it take for OfferNet to build such processes?'

From BT's perspective it is essential that the platform allows partners to discover and use appropriate semantically described services. To achieve this we have carried out the steps described below. Some of these are directly relate to the SOA4All methodology (steps 3-6) whist others (steps 1 & 2) are supporting activities that must be carried out in order to enable benefits of the SOA4All approach to be realised.

1. We have created an RDFS taxonomy describing the nature of currently available telco and telco-related services. The taxonomy is based upon the results of the extensive survey carried out on current activity in the area (see deliverable D8.5). The taxonomy, containing around 80 concepts is available at:

   http://ngwr.labs.bt.com/Ontologies/TelcoAPITaxonomy.rdfs

2. We have created an RDFS ontology describing the data requirements of the APIs of telco services. We have used the Ribbit API as a guide and generated an ontology with 13 classes and 110 properties that describe the domain. This is available at:

   http://ngwr.labs.bt.com/Ontologies/TelcoAPI.rdfs

3. We have created semantic service descriptions for the Ribbit API. Ribbit provides a RESTful API[1] as well as abstractions from the API using various programming languages. We have used the project's SWEET tool to generate MicroWSMO descriptions for all the Ribbit services which refer the domain ontologies. There are 8 Ribbit services with around 40 operations within them that have been described. In addition to this we have also used the Ribbit Java API to create code that we have then wrapped as WSDL services. These WSDL services have been semantically described using SOWER to relate them to the domain ontologies and to generate WSMO-Lite descriptions. In addition, the Grounding Editor has been used to describe the service input and output and used to generate lifting and lowering transformations for data between the semantic and non-semantic levels. The two-pronged approach of using both RESTful services and Java wrapped as WSDL has been necessary since the project's support for WSDL services is far more advanced than that for RESTful services. It is not currently possible to execute RESTful services (other than those using the GET method) using the Execution Engine and Consumption Platform. In addition the wrapping allows us to handle the authentication required for the services more easily

4. We have uploaded semantic descriptions of services to the project's service registry, iServe[2] which, using the WSMO-lite annotations adds descriptions of message parts for the services that are required by the Process Editor.

5. We have annotated 3rd party services. In addition to the Ribbit services, BT must also ensure that there are sufficient services available via the platform to create a critical mass of potential service offerings. In this spirit we have identified the services

[1] http://docs.ribbit.com/restful-api

[2] http://iserve.kmi.open.ac.uk/browser.html

required to realise the processes in the Offers4All scenario and created service descriptions for these. These include support for Facebook and Foursquare interaction and ancillary services such as billing and offer vetting and recommendation. In practice the addition of these services may be requested by the partner and described by BT and then hosted (privately) on the platform for integration into processes.

6. We have executed these services in the SOA4All Studio via the Consumption Platform

From OfferNet's perspective, the aim is to use the platform to generate processes which are then integrated with their front (i.e. customer facing GUIs) and back end systems (i.e. data persistence). To achieve this, we have carried out the following steps. Step 5 directly relates to the SOA4All methodology whilst the other steps are supporting activities.

1. We have developed a set of processes that are required to realise the Offers4All scenario. These processes have been represented as UML sequence diagrams, which although not a requirement of the SOA4All methodology helps in the definition of actors, their roles and the services and infrastructure needed to support them. There are 17 processes in all which identify interaction by offer providers and users and the required response to that interaction. One such sequence diagram is shown in Appendix 1. In addition there are processes dealing with interactions instigated by OfferNet (such as billing and monitoring) which also have a required response.
2. We have developed a domain ontology that describes the Offers4All scenario focussing on the data requirements of the various services involved. The ontology contains about 25 classes and 50 properties and includes the Nepomuk contact ontology[3] which is used to handle the contact information associated with offer providers and users and the W3C Geo ontology[4] which is used to handle location information. This is available at:

   http://ngwr.labs.bt.com/Ontologies/Offers4All.rdfs
3. We have developed front-end infrastructure, which is a GUI allowing offer providers to enter and edit their details, describe offers and track the progress of offers and a GUI allowing users (i.e. offer receivers) to enter their details including preferred communication channels, describe their interests, and browse for and respond to offers. The GUIs have associated Web Applications hosted by Tomcat which call the composed processes of the scenario as WSDL services (see 5. below). Further details are provided in section 3.3
4. We have developed back-end infrastructure, which is a set of APIs and associated persistence to enable the creation, deletion and editing of user data (for both offer providers and offer receivers), offer data and consumption data i.e. which users have interacted with which offers. These are RESTful services which conform to the Offers4All ontology. The persistence is provided by an OWLIM triplestore instance. The project's Semantic Spaces, which are public, have not been used here since the data is private to OfferNet. There are three APIs with associated storage. One stores user details (both offer provider and end-users) including contact details, location, preferences, etc. A second stores offer details i.e. a descriptions of the offers including what kinds of users they are targeted at. Finally, a third is a consumption API that records which users have consumed which offers or forwarded offers to

[3] http://www.semanticdesktop.org/ontologies/nco/

[4] http://www.w3.org/2003/01/geo/

others. Further details are provided in Section 3.4

5. We have developed a set of processes which satisfy the requirements identified in the sequence diagrams described in Figure 1. These processes have been initially developed using Java to handle the control and data flow between services and then exposed as WSDL services which consume and produce RDF instance data, allowing them to be called by the front-end infrastructure. Alongside the Java processes we have also developed processes using SOA4All technology as this has become available to meet the needs of the case study. This involves building processes using the Process Editor, binding activities in the process to service operations described by semantic annotations in iServe, defining the data flow in the process using the message part descriptions associated with the service annotations and finally deploying the process to the Execution Engine resulting in composed WSDL services using semantic I/O which can then be called by the front-end infrastructure. The Java-based processes allow us to illustrate the required functionality of the scenario but will be replaced by those developed using the Process Editor as the capabilities for process deployment and execution increase. Further details are provided in Section 3.5
6. We have built an application for Android phones. The application allows users to browse and act upon offers that are relevant to them. It interacts with the processes to get details of offers and to launch appropriate communication with Ribbit services when users choose to do so. The Android app allows us to illustrate how mobile users can interact with the Offers4All service, meeting their location-based requirements. Further details are provided in Section 3.6

An architecture of the prototype is given in Figure 1 below. It illustrates the major components of the Offers4All service and the BT and 3rd party services that are necessary to support it. The left and right blocks of the architecture show the front-end infrastructure of the service supporting the offer providers and users. The top block shows examples of Ribbit, Offers4All and 3rd party services that are used to create the processes. The bottom block illustrates the back-end persistence required by the service including the triple store and the various storage APIs. Finally, the centre block shows the processes required by Offers4All. These components rely on SOA4All technology, from the use of SWEET and SOWER to semantically describe the services, the iServe repository to store these descriptions allowing them to be discovered, SPICES, the consumption platform, allowing the services to be found, executed and rated, the Process Editor allowing services to be composed and the Execution Engine, allowing composed services to be executed. The architecture also illustrates how the SOA4All provided elements fit with those that are outside the direct scope of the project but nevertheless have been constructed for the scenario in order to allow the full benefits of the SOA4All approach to be shown i.e. the storage APIs and infrastructure and the user interfaces.

A table providing full details of the usage of SOA4All technology is provided in Appendix 2.

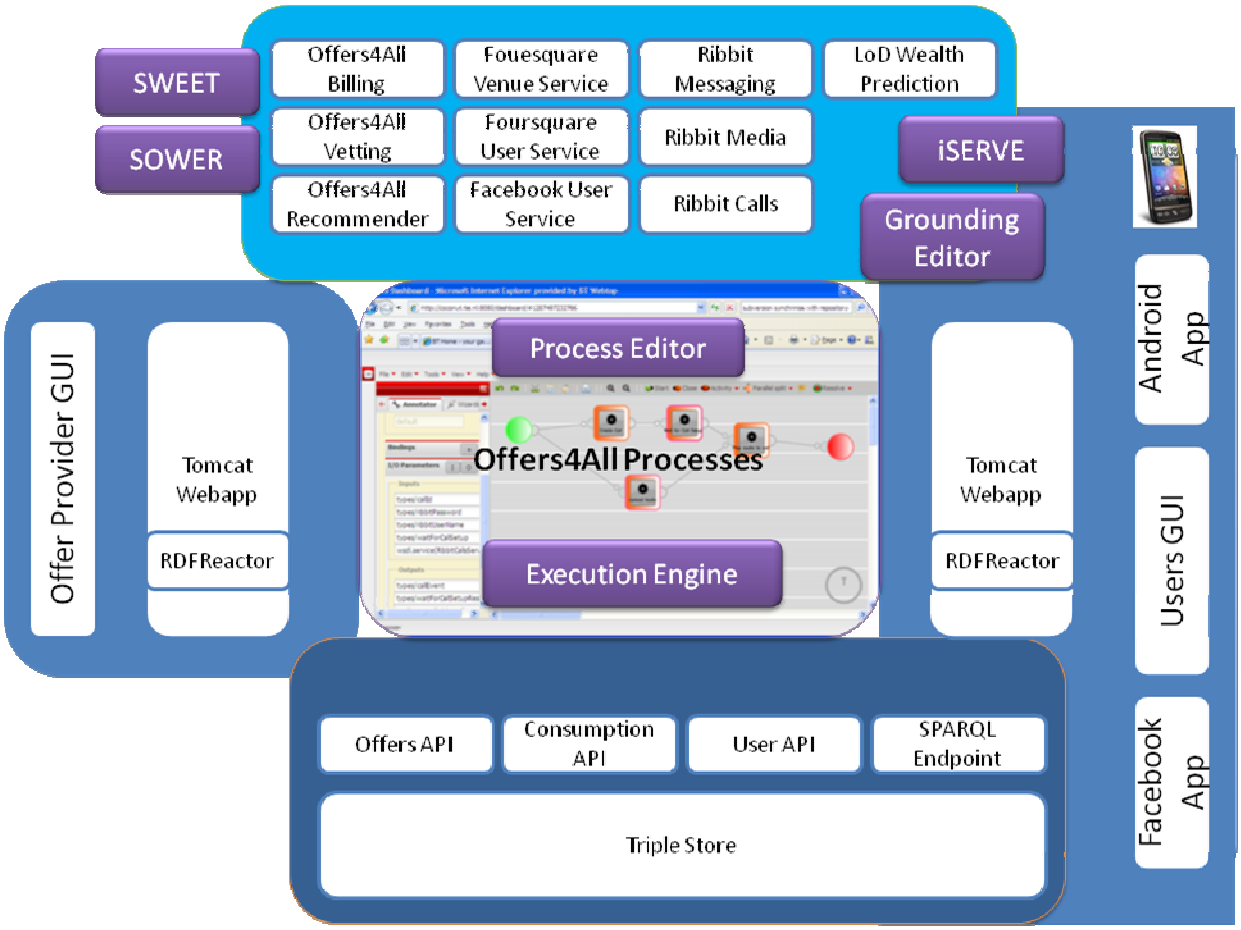


Figure 1. Offers4All Architecture

3.1 Annotation and Storage of Services

BT, as the provider of a platform based on SOA4All technology must ensure that there is a rich set of services available for users of the platform to build novel mashups and processes. Each service must be annotated such that it can be used by the design-time tools of SOA4All. WSDL services can be annotated using SOWER, the WSMO-Lite editor, which allows ontological annotations to be inserted into the WSDL using SAWSDL model references. Figure 2 shows an example of a WSDL service – the RibbitCalls service being annotated in SOWER. The WDSL description is shown on the right-hand-side. One or more ontologies can be opened in the left-hand panel and concepts within the ontologies can be dragged to elements of the WSDL resulting in an annotation being made. In the Figure the User concept from the TelcoAPI ontology has been associated with ribbitUserName which is an input type of the 'playMediaToCall' method in the RibbitCalls service. This creates the sawsdl model reference as shown at the bottom of the Figure.

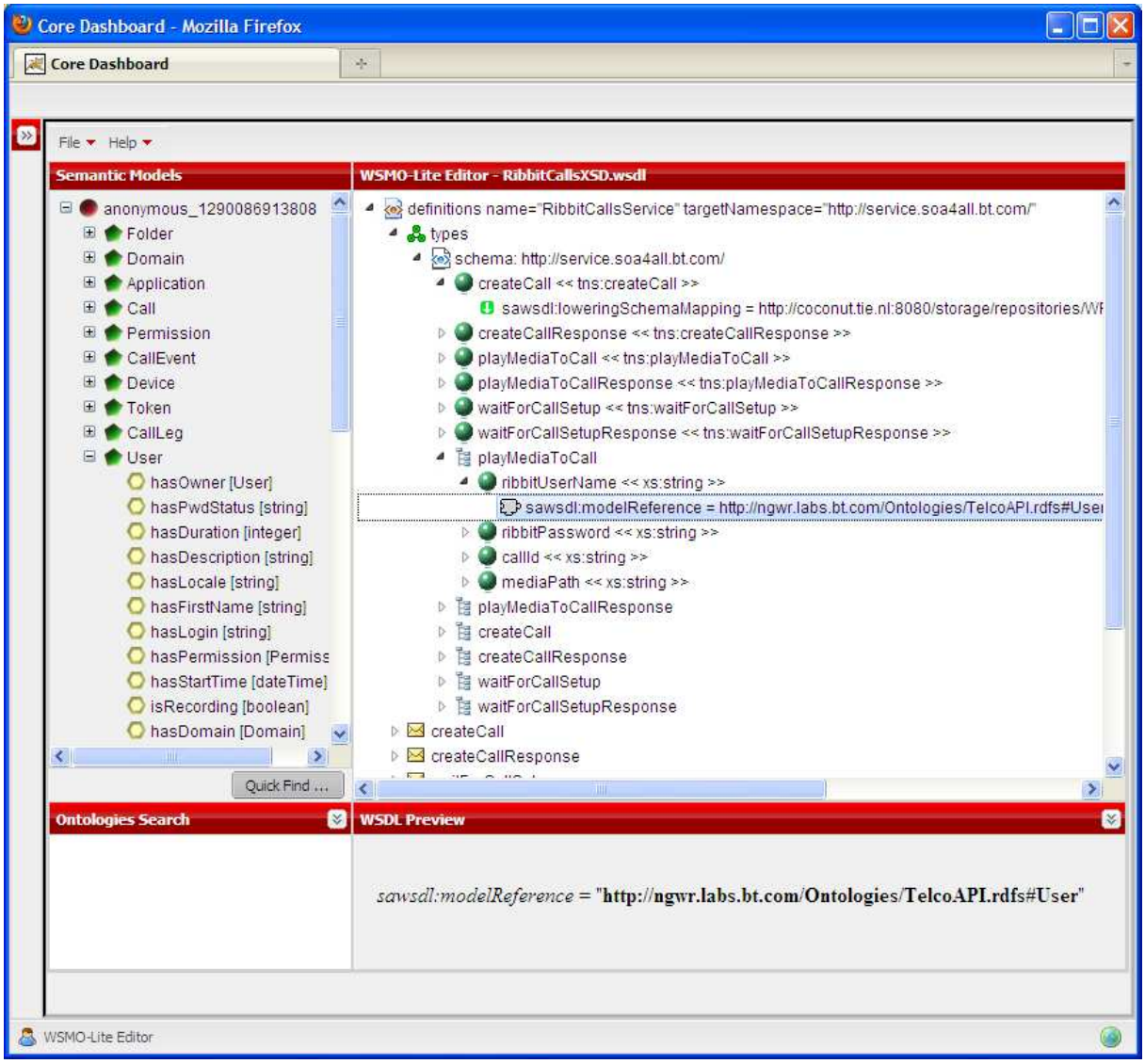


Figure 2. The Ribbit Calls WSDL undergoing annotation in SOWER

Annotations on Restful services are carried out using the SWEET tool as shown in Deliverable 8.4 which describes the first prototype. This results in a MicroWSMO description of the service.

For both WSDL and Restful services it is necessary to generate lowering and lifting transformations. This allows RDF instance data (defining the inputs to a service) to be lowered to XML and service output data expressed as XML to be lifted back to an RDF format.

Figure 3 shows a screenshot from the Grounding Editor where the RibbitCalls service is being mapped to the TelcoAPI ontology. The 'ribbitUserName' input of the playMediaToCall operation is shown being related to the hasLogin property of the User concept in the ontology. Once a full set of mappings have been made the lifting and lowering transformations can be generated as XSLT (XML stylesheet transformation) files. In SOWER, these XSLT files can be related to the WSDL descriptions, again using a SAWSDL model reference. The complete annotated descriptions can then be uploaded to the iServe repository which is able to process them and generate the message partonomy so they can be used in the Process Editor.

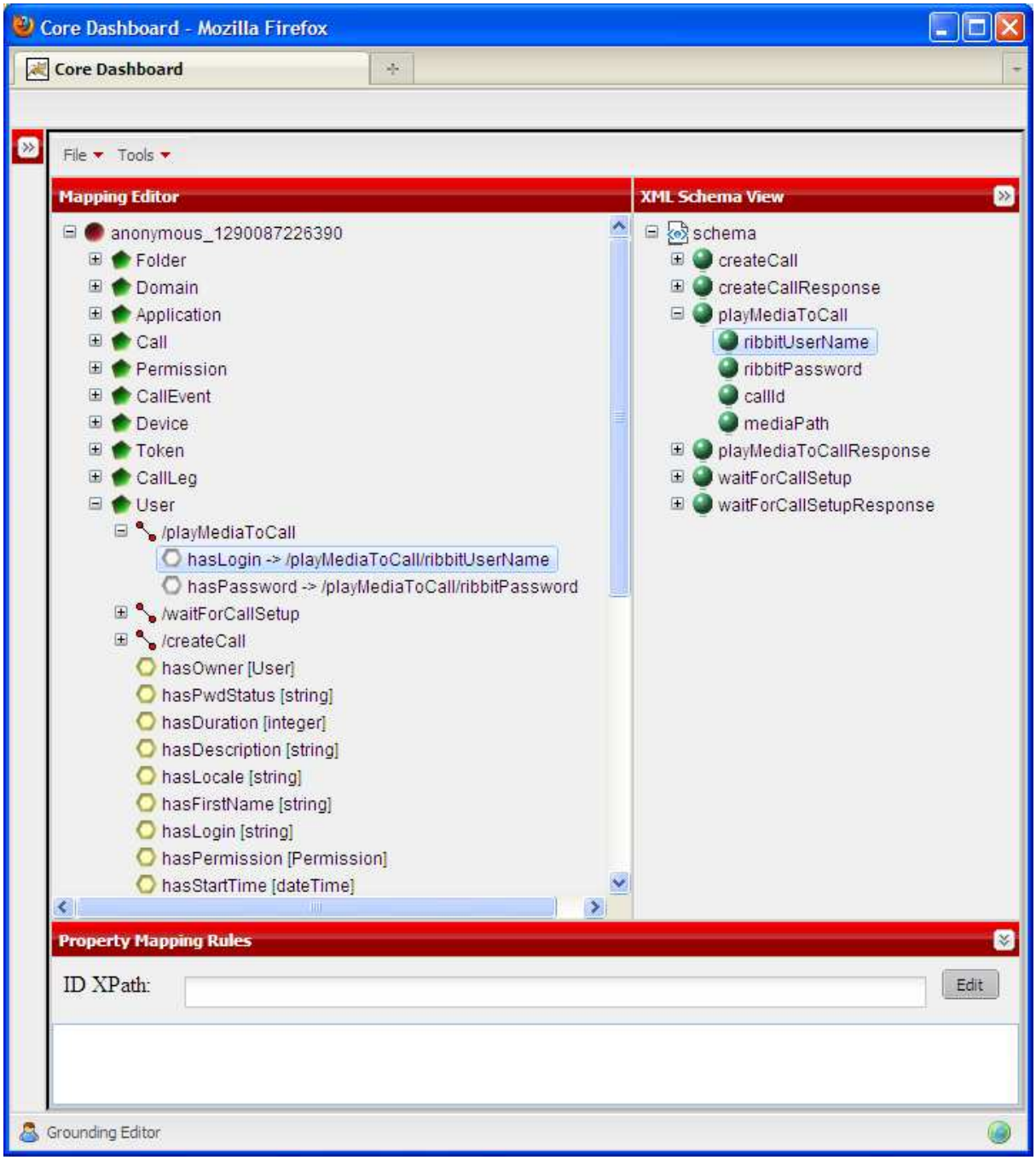


Figure 3. Using the grounding editor to create mappings between domain ontology and XML schema.

3.2 Consumption of Services

The SPICES consumption platform allows individual services to be found, executed and added to a list of favourite services and operations. Figure 4 shows a screenshot from SPICES. The left-hand panel allows the user to perform various actions such as searching for services, browsing by service categorisation, managing favourites and accessing recommendations. Results of the actions are shown in the right-hand panel. In the screenshot the 'Ribbit Send SMS' service has been selected from the list of favourites. A form is created on the right-hand-side allowing the user to enter the required inputs and execute the service. The favourites list is available throughout the SOA4All Studio allowing

favourite services to be selected within other tools such as the Process Editor.

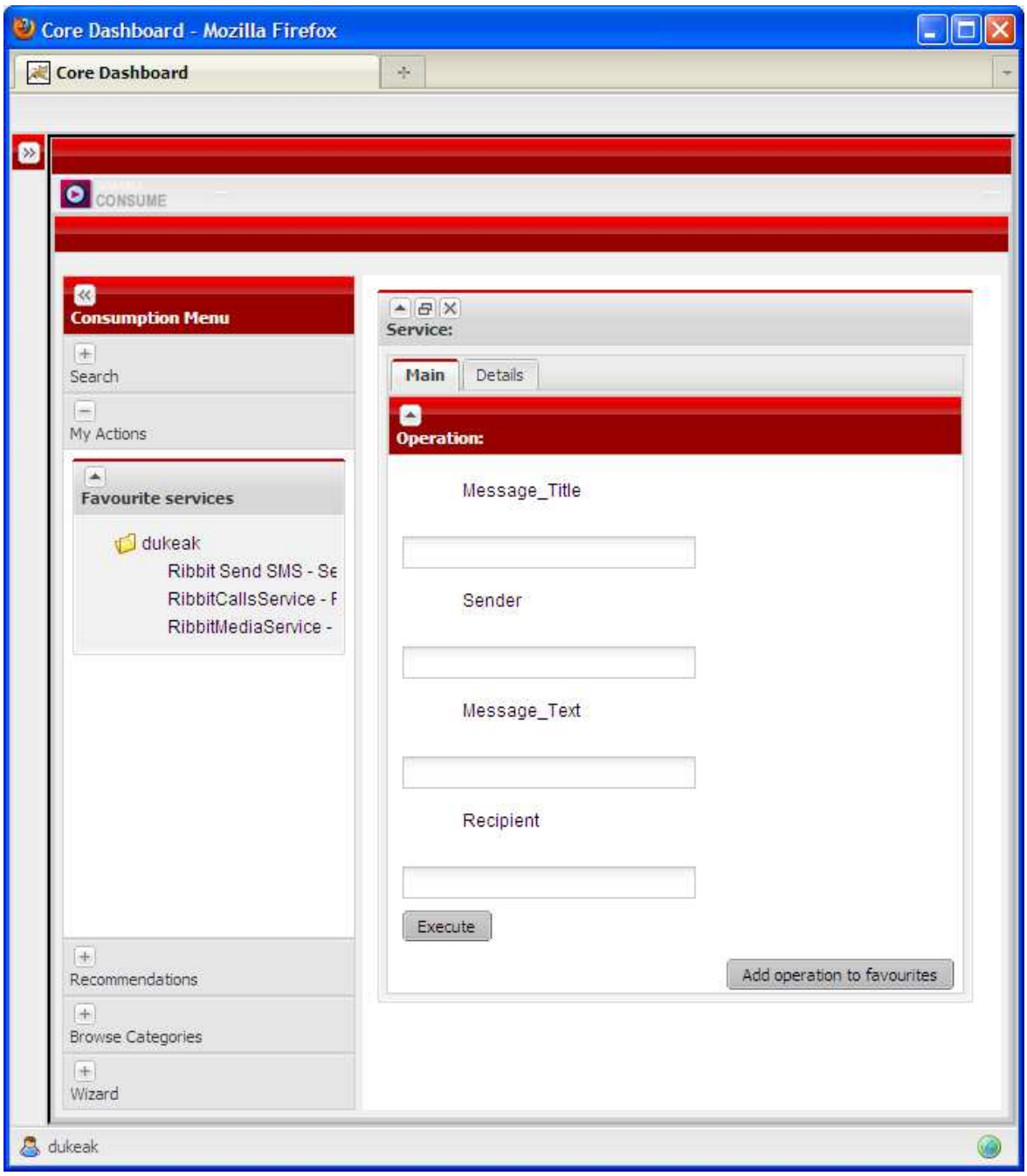


Figure 4. SPICES Consumption Platform

3.3 Offer Provider Front-end

The prototype includes a front-end, which incorporates a graphical user interface and an associated Web Application that illustrates how offer providers can enter new offers and manage existing offers. The front-end communicates with the processes that are defined by OfferNet using SOA4All technology. In practice, companies such as OfferNet would need to either build such interfaces and connect them to processes or adapt existing interfaces to do so. In SOA4All, processes built using the Process Editor are exposed as WSDL services in their own right, hence the Web Application is able to call these WSDL endpoints and handle the response, reflecting this in the GUI.

The front-end has been built using the Google Web Toolkit[5] which provides facilities for building browser-based applications. This interacts with a Java-based Web application. Both are hosted on Apache Tomcat.

Figure 5, shows a screenshot from the offer provider front-end. Here an offer provider is shown their current offers and their contact and location details. Upon selecting an offer they are shown the detail of that offer which can be seen in the screenshot in Figure 6.

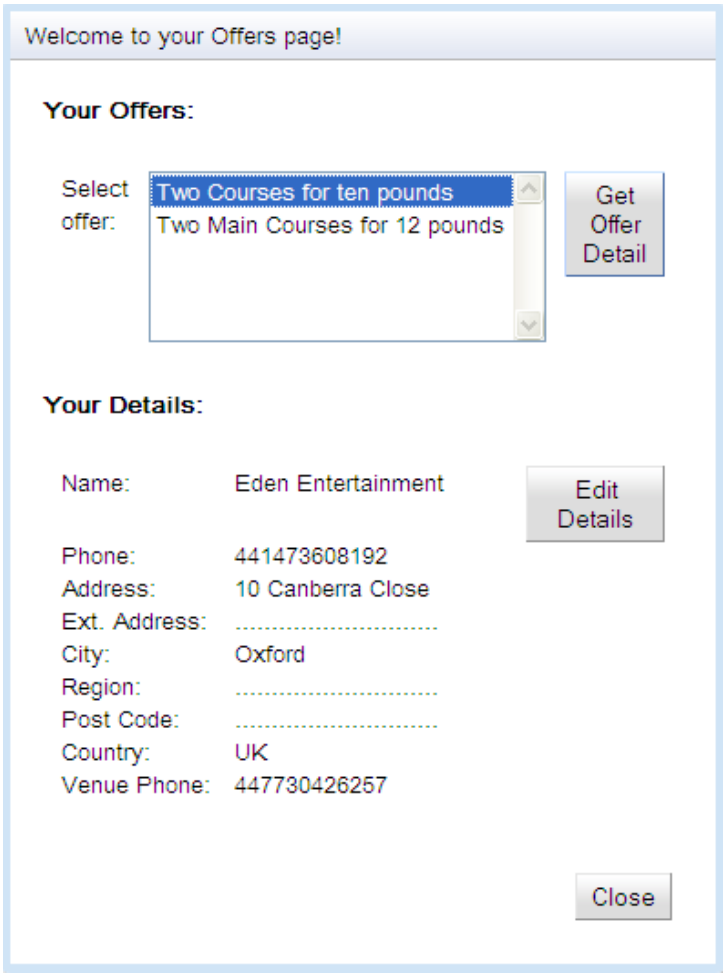


Figure 5. Screenshot from offer provider portal.

In Figure 6, the offer provider is shown changing the status of the offer from 'Prelaunch' to 'Launched'. This action results in a back-end processes being executed which identifies the most relevant end-users to send the offer to, finds the contact details for those people and then initiates the appropriate communication service to notify them of the offer.

[5] http://code.google.com/webtoolkit/

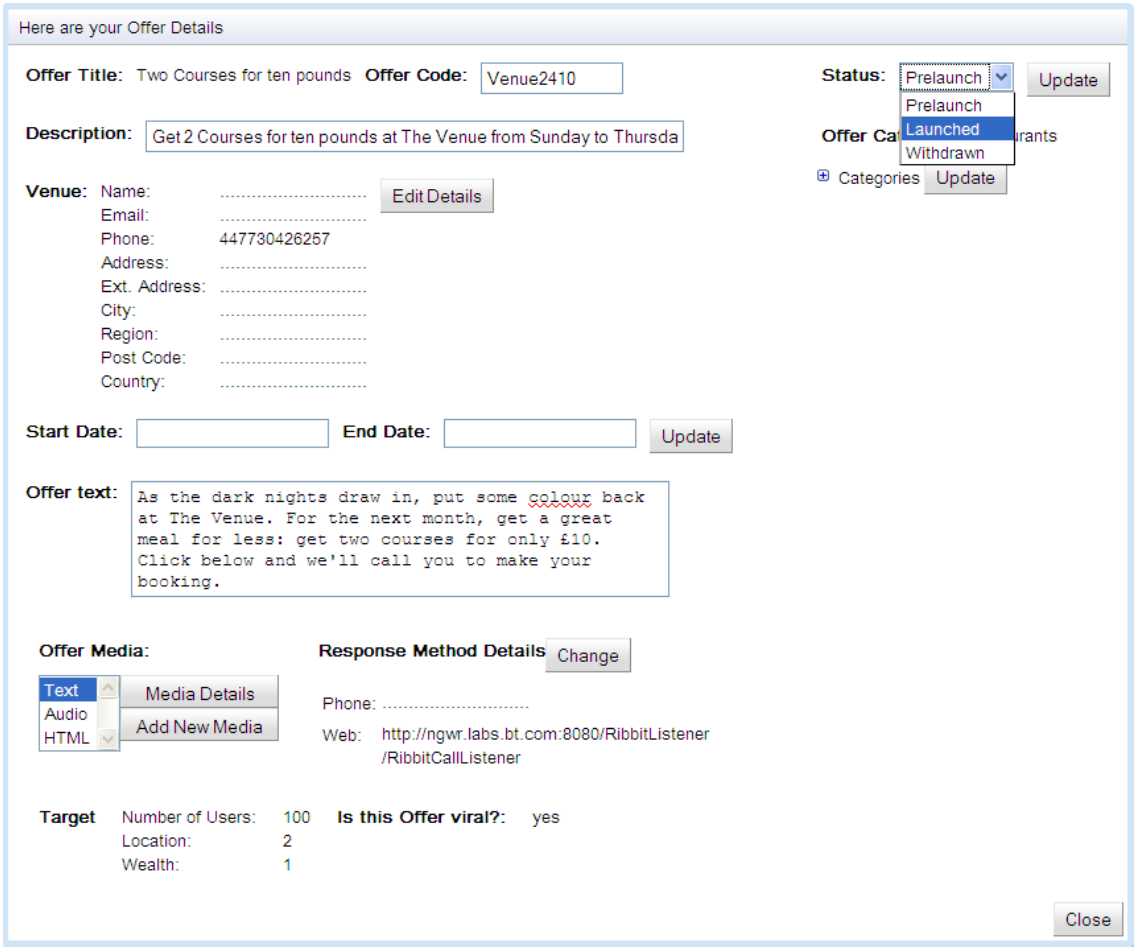


Figure 6. Screenshot showing details of offer

3.4 Storage APIs

The data used in this prototype is stored in a triple store, accessible through a SPARQL endpoint and through a set of specific RESTful APIs tailored for read-write access and manipulation of concrete parts of the data. There are three such APIs in the system: Users API, Offers API, and Consumption API.

Each of the APIs is a set of linked resources (hypertext, RESTful), with inputs and outputs mostly in RDF. To illustrate, the following is an operation-oriented view on a part of the Users API, listing the functionalities available for managing users and how they map to the resources of the service. In the list, the words GET, POST, PUT and DELETE are HTTP methods available on the following URIs; and terms of the form uc:* are classes and properties of the use case ontology.

- listUsers – GET `/users` – returns RDF that lists the instances of uc:User as dereferencable pointers of the form `/users/{id}`
- addUser(instance-based triples) – POST `/users` – adds a user; the input should contain a single blank node of type uc:User which describes the user to be added (using triples about that node); the method returns the newly-assigned identifier for the instance
- getUser(id) – GET `/users/{id}` – returns RDF that describes the user; plus it

includes graphs (sets of reified[6] statements, further discussed below) that point out where to add new properties and where to delete existing ones;

- addToUser(id, property-based triples) – POST `/users/{id}` – allows adding any kind of RDF property to the user record, the input must have triples that originate from the user instance identifier
- getUserLikes(id) – GET `/users/{id}/likes` – returns RDF that contains all the uc:likes statements ("user likes an offer") about the given user (plus their reification graphs to indicate where each statement can be deleted)
- addUserLike(id, uri) – POST `/users/{id}/likes` – adds a uc:likes statement and returns its reification to indicate where it can be deleted
- deleteAllUserLikes(id) – DELETE `/users/{id}/likes` – removes all uc:likes statements
- deleteUserLike(like-id) – DELETE `/users/{id}/likes/{like-id}` – removes a single uc:likes statement (the URI with *like-id* is known from the reifications above)
- Dislike – the same four operations above for uc:dislike statements about offers the user dislikes
- Interest – the same four operations for uc:hasInterest statements
- Disinterest – the same four operations for uc:hasDisinterest statements
- Contact – the same four operations for uc:hasContact statements
- deleteUserValue(v-id) – DELETE `/users/{id}/v/{val-id}` – removes a single statement other than those specially identified above; the *val-id* is known from reifications returned by getUser(id)

For offer providers (instances of uc:OfferProvider), also managed by the Users API, the API contains the same type of operations but focusing on uc:hasOffer, uc:hasContact, uc:hasVenue and uc:hasDefaultVenue statements (no liking, no interests). The Offers API has the same structure, working with instances of uc:Offer and its properties (e.g. uc:description, uc:hasOfferCode, uc:hasOfferStatus etc.), and the Consumption API deals with offer forwards, offer recommendations, offers sent to users, and user responses to offers, again with the same structure of operations.

Figure 7 explains the structure of the data managed by the APIs, especially including the reification graphs for manipulating the statements. When a new user record is submitted to `/users`, a new ID is assigned to it, in the Figure `/users/1345`. (Note that the Figure assigns the identifier `/users/1345#this` to the instance and `/users/1345` to the enclosing graph, so as to avoid confusion between the users and the documents that describe them; however, this distinction is not yet implemented in the system).

In Figure 7, the following information was submitted about a new user:

```
_:x a uc:User;
    uc:likes ex:Amazon, ex:Google;
    uc:hasInterest ex:Cars;
    uc:hasWealth "56".
```

On the user graph resource (`/users/1345`), a client can invoke GET to retrieve all the data, POST to add new properties about this instance, and DELETE to remove the user.

[6] A "reified statement" is a set of statements that describes this statement. For example, for a statement that A Likes B, its reification is "a statement whose subject is A, whose predicate is Likes, and whose object is B". We use the term "reification graph" later as a graph that contains statements.

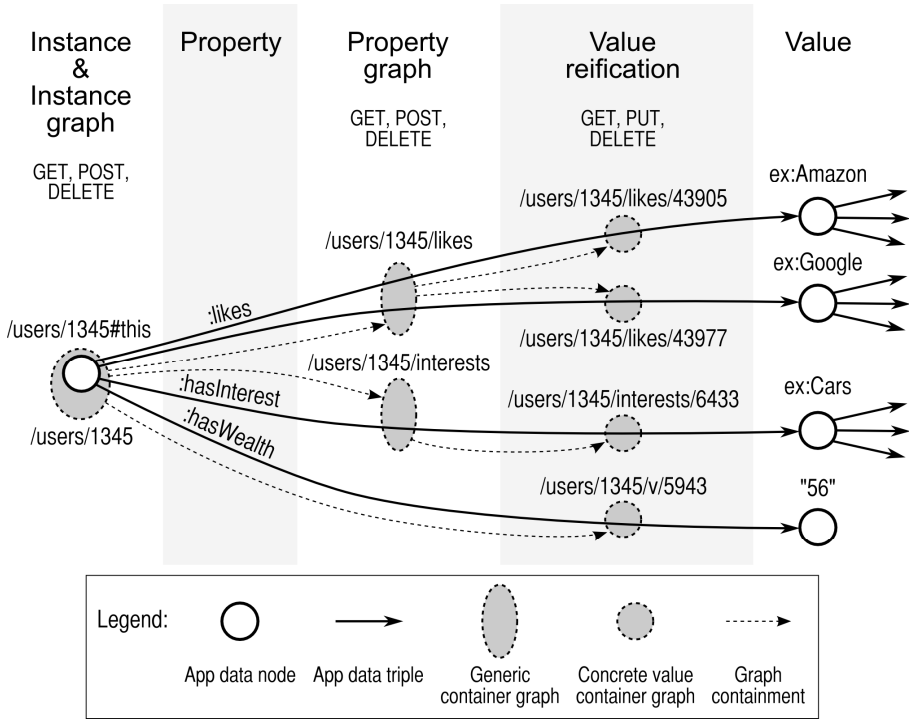


Figure 7. Structure of data managed by the storage APIs

For properties that the system is configured to call out (for users it's the properties uc:likes, uc:hasInterest etc. as listed earlier), it creates graphs such as `/users/1345/likes` and `/users/1345/interests`, where the client can invoke GET to retrieve all the statements of that particular property, POST to add a value with the given property to the instance, and DELETE to drop all the values for the given property (currently not implemented).

For every particular statement (such as `_:x uc:likes ex:Amazon`), the system creates a reification graph like `/users/1345/likes/43905` that the client can GET to retrieve that particular statement (but that's not particularly useful), DELETE to remove that statement from the system, and PUT to replace the value (not currently implemented).

For statements that do not use the specifically called-out properties (here `_:x uc:hasWealth "56"` is an example of such a statement), the reification graphs have URIs like `/users/1345/v/5943`, where the client can again invoke GET, DELETE and PUT as above.

The various graphs are all linked: the instance graph *contains* the property graphs and the `/v/` value reification graphs, and the property graphs *contain* the called-out value reification graphs. Every value reification graph contains a single reification with the triple of interest.

In order to replace some triple (say `/users/1345 :likes ex:Amazon`), the client would DELETE the value graph that contains the reification of this statement (in this case `/users/1345/likes/43905`) and then POST the new value to `/users/1345/likes` (or post the whole triple to `/users/1345`). The return value of this POST will tell the client the URI of the new value reification graph.

The software used to implement all these APIs is a configurable triple-store wrapper built in Java with the Jersey framework. The wrapper implements the API resource (`/users`), the instance graphs (`/users/{id}`), the property graphs (`/users/{id}/{prop-name}`) and the value reifications (`/users/{id}/{prop-name-or-'v'}/{val-id}`) as four different types of resources. We plan to continue work on this software as we expect it to be very useful also

outside the use case.

3.5 Offers4All Processes

As stated above, in building the prototype, we have used an approach where we initially produced composite services by using Java to manage the calls to individual WSDL or Restful services and the flow of data between them. This allowed us to build the required functionality of the prototype fairly quickly. Following this we have replicated these processes using the Process Editor (PE) developed in Workpackage 6. One such process that has been built using the PE is shown in Figure 8. This process, which enables an audio file to be played over a phone call, uses four separate operations on the Ribbit Calls and Ribbit Media services. The operations are as follows:

RibbitMedia:UploadMedia – Uploads an audio file to a specified folder on the Ribbit platform

RibbitCalls:CreateCall – Creates a call between two or more people

RibbitCalls:WaitForCallSetup – Waits for the call to be established

RibbitCalls:PlayAudioToCall – Plays the audio file to the call

The intention is that this process will be used by offer providers to test offer related media over a call prior to the offer being launched. It will allow them to ensure that the audio describing their offer is available and of sufficient quality when played over the call. This test facility is provided in the offer provider GUI shown in Figure 6 and results in the process described here being executed.

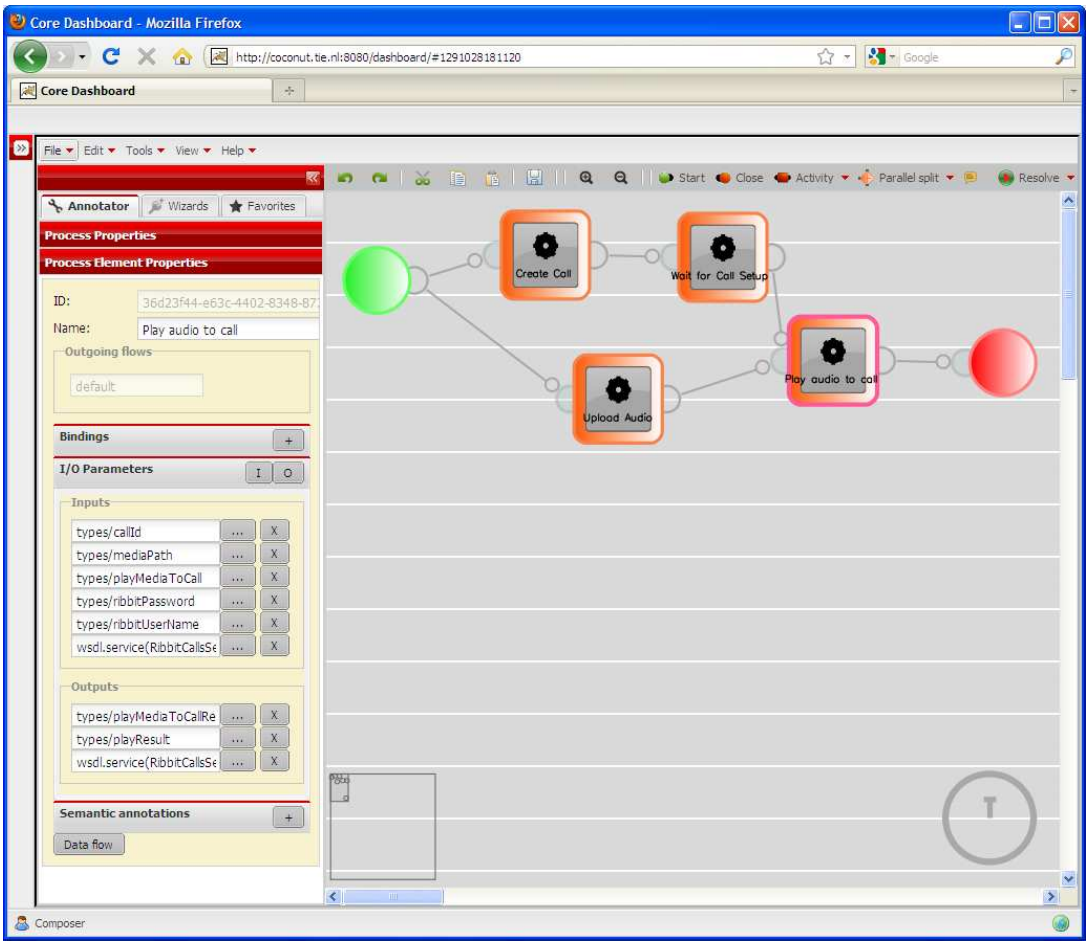


Figure 8. 'Play audio to call' process shown in the Process Editor

In Figure 8, the process with four activities is shown. Each of these activities can be bound to a particular operation of a WSDL service or a RESTful API call. The user is able to perform the binding by entering details of where the semantic annotations on these operations are stored in the iServe repository or they can simply select from the list of favourites provided by the consumption platform. Figure 8 shows the binding for the 'Play audio to call' activity which is the right-most activity in the process. The activity is bound to the 'RibbitCallsService – play media' operation as can be seen in the panel on the left. The inputs and outputs for that operation which are determined from the annotations are also shown.

Once the activities have been bound, dataflow in the process can be carried out. Figure 9 shows a screenshot of the data flow editor for the 'Wait for Call Setup' activity. The 'Inputs' panel shows the inputs to the activity. The 'callid' input is highlighted indicating that this is the input for which the data flow is being defined. The 'Available outputs' panel shows the data outputs of preceding activities (or the start activity indicating data that is input to the composed process) which can be selected and associated with the current input by dragging them into the 'Associated outputs' panel. In Figure 9, the output 'callid' from the 'Create Call' activity has been dragged to the 'Associated outputs' panel and is thus associated with the 'callid' input to the 'Wait for Call Setup' activity. The data flow itself is carried out by a SPARQL CONSTRUCT query which can be generated by clicking on the 'Generate' button in the 'Data transformation' panel. The SPARQL query for the callid input is shown in Figure 9. Alternatively, this SPARQL can be generated automatically for the whole process when the 'Deploy' button in the PE is clicked.

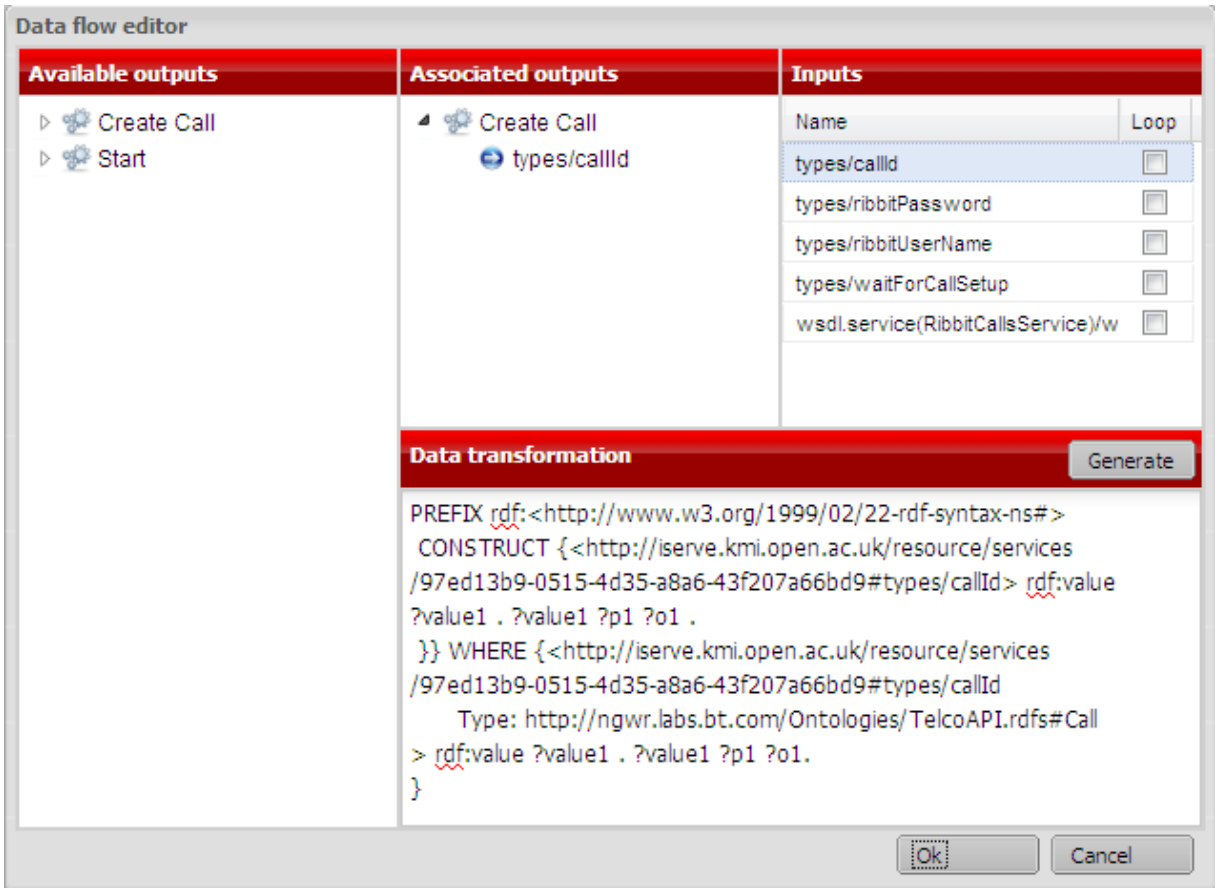


Figure 9. Data flow editor

3.6 Offers4All Android Application

This section discusses the development of the "Offers4All Android application", which has

been created as a showcase of the potential that SOA4All technologies have in different environments other than the browser. In particular, it shows how end users are able to interact with the Offers4All service and how the back end infrastructure of the Offers4All service is easily employed and reused in a variety of settings. We have focussed on the increasingly popular Android Smartphone but similar solutions could easily be created for iPhone/iPad, Blackberry or mobile browsers.

Regardless of the application environment chosen, the idea is that a simple mobile application is able to feature a wide range of functionalities by leveraging the underlying SOA4All technologies. In other words, even if the code of the application itself is very limited, by interacting with SOA4All components, a powerful application can be presented to the end-users.

The Android App allows the user choose from a set of categories, thus indicating the types of offers that they are interested in, via a simple menu with checkboxes. Then, based on the selected categories, and the location of the user, the application connects to the Offers4All backend, which returns a set of suitable offers. Upon selection of an offer, the full information for that offer is displayed. The user can opt to consume the offer via the app by a variety of means. They can request that the offer provider calls them, visit a webpage allowing them to consume the offer or they can choose to forward details of the offer to a friend via SMS. Each of the actions is performed by initiating a process via the back end infrastructure. For example, if they prefer that the offer provider calls them a process is initiated that creates a call via the Ribbit Calls service and then records the consumption of the offer by that particular user via the consumption API.

Figure 10 depicts the described functionality by the navigation between the different menus (“Activities”, in Android programming terminology). From the main entry point (labelled 1), the user can select the profile/categories activity (2) as well as the list of available offers (3). Upon selection of a particular offer, a full description is presented (4), from where different actions can be triggered, such as receiving a call from the offer provider (5). Arguably, the application itself is not complex. The major functionality is provided by the Offers4All processes provided by the SOA4All-powered infrastructure.

Figure 10. “Offers4All Android app” navigation menus

As explained before, the application installed in the phone (or Android device) communicates with the backend through its APIs (see Figure 1). There are two main communication points between the application and the SOA4All-powered Offers4All backend:

1. The selection of the offers relevant for the user, meeting both the location-based constraints and his particular preferences, is performed in (3) by querying the Offers API. The backend returns a list of suitable offers given the constraints.
2. When a user selects one of the available actions related to an offer, a process is initiated. Depending on the type of action, the user is required to provide more information (for example, if the process to be launched includes a text message to be sent) or not (for example, if the action implies a call to be received by the user). The process triggers a communication event e.g. a call or message via Ribbit and stores the event in the consumption API e.g. for billing purposes.

It is worth noting that including a different set of consumable actions into the application would be very easy from the app point of view, as it would just imply linking to different services/processes of the Offers4All service. Thus, by using SOA4All infrastructure, the versatility of such an application is greatly increased.

4. Next steps

We will continue to develop the Offers4All prototype as an example of a many-sided business benefitting from the use of SOA4All technology.

We plan to develop a Facebook application to allow users to be made aware of and to respond to relevant offers whilst they are using Facebook. This will illustrate how Offers4All can support users who use social networking as their principle means of online communication. The app will also interact with the back-end processes described above, showing how they can be reconfigured to deal with new communication channels.

We also plan to develop a back-end service using Linked Open Data provided by the UK Government. This will use census data linking geographic location with earnings and will exist as a service allowing OfferNet to predict earnings based upon a user's postcode which can then be used to improve the targeting of offers. The service will illustrate how Linked Services providing data can be integrated into service mashups such as Offers4All alongside more traditional services.

We will continue to interact with the technology workpackages to ensure that our requirement to support the use of RESTful services are met and that all Offers4All processes are implemented using SOA4All technology.

In the remainder of the project we will evaluate the SOA4All federated infrastructure by using it to deploy the Offers4All service. We will align our work with BT Strategy's work on Communications-enabled Business Processes. A traditional software development process is underway to support a user trial in that project. With BT Strategy, we will evaluate the technology against requirements that emerge from the trial with the aim of showing how SOA4All technology could enhance take-up. Further exploitation routes such as Ribbit and BT.com will also be considered.

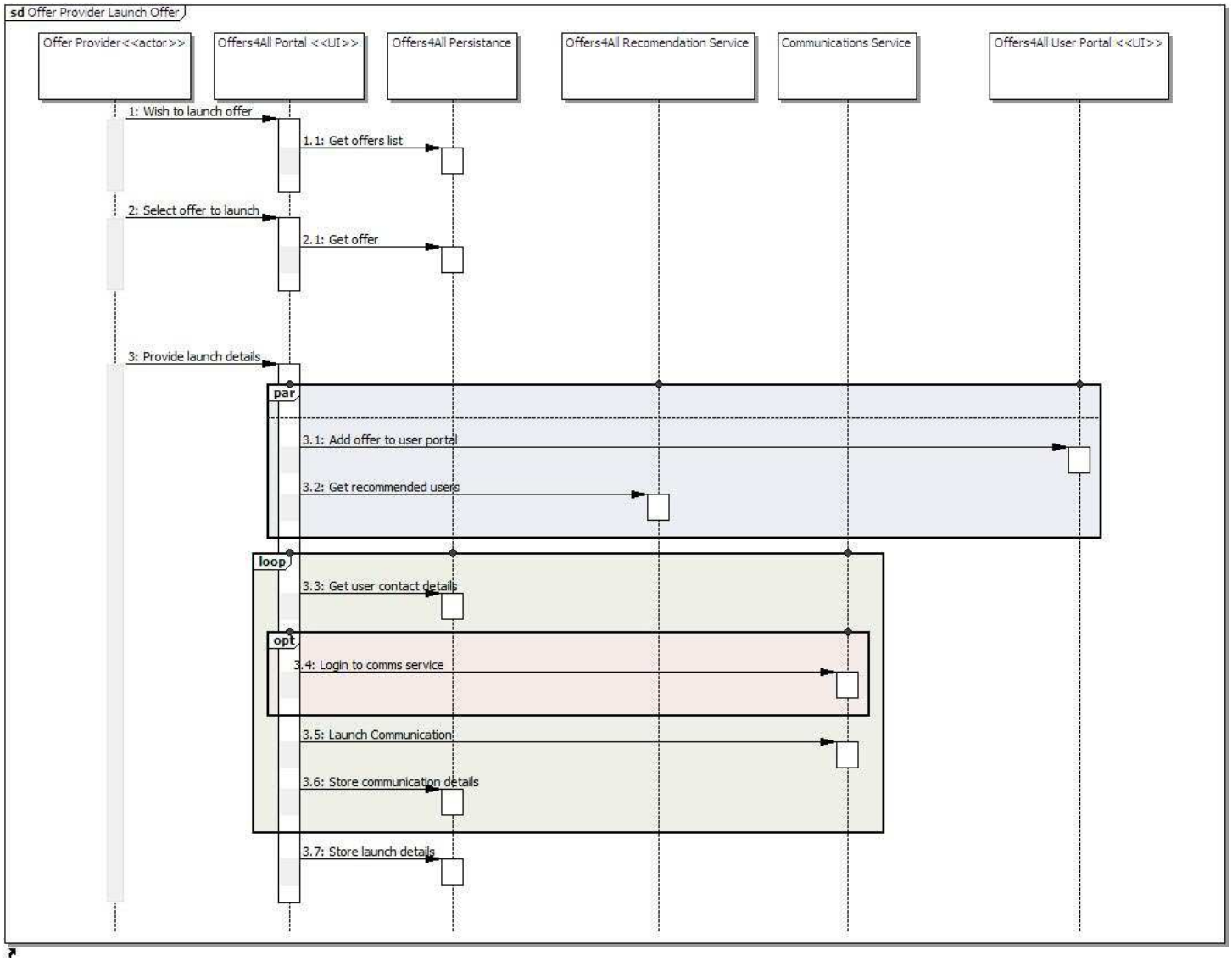
We will carry-out a usability evaluation of the SOA4All tools and approach within BT in collaboration with the University of Manchester. This will be reported in D8.7 and will compare SOA4All tools in the context of the BT case study with similar tools for service annotation and consumption and process construction. This will allow us to backup the claims made by the project and discussed in Section 2.1of this Deliverable.

5. Conclusion

In this deliverable, we have described the 2nd prototype of the BT Case Study which is based on the Offers4All scenario. The deliverable describes how such a scenario can be realised by the creation of a BT platform based on SOA4All technology. The activities required that enable the platform and the realisation of the Offers4All service upon it have been described. This involves the use of SOA4All technology (for annotation, consumption, process creation and execution) but also other activities that must be carried out alongside this such as ontology creation and front-end and back-end infrastructure development.

The Offers4All scenario has been developed as an example of a many-sided business model. Such models which include communication at the heart of their business processes can benefit from an advanced platform provision such as that developed in SOA4All. We will continue to adopt and evaluate the technology, aligning it with potential exploitation routes within BT.

Appendix 1 - Example of Offers4All Sequence Diagram



Appendix 2 – Use of SOA4All Components

WP	Component	Usage
1	DSB	Deploy Ribbit and Offers4All services
2	iServe	Storage of service descriptions
2	SOWER	Annotation of WSDL services
2	SWEET	Annotation of REST services
2	Recommender	
2	SPICES	Execution of single services, adding to favourites
2	Composer	Creation of Offers4All processes
2	Monitoring	Planned inclusion at M36
2	Storage Service	service ontologies, services favorites
2	Authentication	
3	Data Grounding Editor	LiLo schemas for data I/O
3	Reasoner	Used by DTC
5	Crawler & Registry	
5	Discovery	Used by DTC
5	Ranking and Selection	
6	DTC	Planned inclusion at M36
6	Optimizer	Planned inclusion at M36
6	Template Generator	
6	Execution Engine	Executing Offers4All processes
6	LPML API	Used by Process Editor
8	Offer Provider Portal	Allows offer providers to add / edit offers
8	User Portal	Allows users to edit profile and browse offers
8	Offers4All Persistence	Storage of offer data, user data and offer consumption data
8	Android app	Allows users to edit profile and browse offers