Sensoria

016004
Software Engineering for Service-Oriented Overlay Computers

D8.1.a: Telecommunication Case Study

Requirements modelling and analysis of selected scenarios

Lead contractor for deliverable: [TILab]

Author(s): [Corrado Moiso (TILab), Laura Ferrari (TILab), Ermes Thuegaz (TILab), Marzia Buscemi (UNIPI), Ugo Montanari (UNIPI), Piergiorgio Bertoli (ITC Trento), Marco Pistore (ITC Trento), Raman Kazhamiakin (ITC Trento), Roberto Bruni (UNIPI), Stefania Gnesi (ISTI-CNR), Maurice ter Beek (ISTI-CNR), Marinella Petrocchi (ISTI-CNR), Franco Mazzanti (ISTI-CNR), Jose Fiadeiro (ULEICES) , Laura Bocchi (ULEICES)]

Due date of deliverable: [February 28, 2007]

Actual submission date: [February 28, 2007]
Revision: [Final]
Classification: [CO]

Contract Start Date: September 1, 2005 Duration: 48 months

Project Coordinator: LMU

Partners: LMU, UNITN, ULEICES, UWARSAW, DTU,
PISA, DSIUF, UNIBO, ISTI, FFCUL, UEDIN, ATX,
TILab, FAST, BUTE, S&N, LSS-Imperial, LSS-UCL, MIP, ATXT

Integrated Project funded by the
European Community under the
"Information Society Technologies"
Programme (2002—2006)

Executive Summary

The main objective of the Telecommunication Case Study in SENSORIA is to test and verify the applicability of the insights, methodologies and tools developed in the SENSORIA Project WPs more focused on research activities, on the evolution of the Service Layer of a telecommunication infrastructure towards IT technologies. This document provides the description of the Telecommunication Case Study for the SENSORIA project, and it follows the Sensoria Deliverable D8.0: Case studies scenario description. Moreover, it reports the preliminary results of the application of methodologies and tools developed in SENSORIA to some of the issues identified in the Case Study.

Contents

1 Introduction

This document provides the description of the Telecommunication Case Study for the SENSORIA project, and it follows the Sensoria Deliverable D8.0: Case Studies scenario description. The main objective of the case study is to test and verify the applicability of the insights, methodologies and tools developed in the SENSORIA Project WPs more focused on research activities, on the evolution of the Service Layer of a telecommunication infrastructure towards IT technologies. Moreover, it reports the preliminary results of the application of methodologies and tools developed in SENSORIA to some of the issues identified in the Case Study.

In particular, the Telecommunication Case Study scenario is focusing on the development of applications combining two global computing infrastructures, namely the Internet and Next Generation Telecommunication Networks. In such a context, the applications would be designed, deployed, and executed within a SOA framework that integrates Web Services based computing and a rich set of telecommunication services and capabilities, including call/session control, messaging features, presence and location features, etc. The applications could also be deployed in administrative domains external to the Network Operator, according to the Service Broker business model.

It is important to point out that the proposed Telecommunication Case Study mainly addresses issues concerning the evolution of the service infrastructure/platform, and not specific telecommunication services. Some specific services will be used in the description only to exemplify the infrastructural issues.

This contribution is structured in the following parts: Section 2 introduces the context of the case study, while Sections 3 to 8 provide an overview of the different case study issues: each of the sections describing the case study issues is organised in a description of the issue and of its context in the telecommunication service layer, an exemplification of the issue through one or more use cases, and a presentation of the first results of the SENSORIA research activities.

In particular, the deliverable presents the first results of the research activities of the other WPs of SENSORIA on composition mechanisms, taking into account asynchronous/synchronous orchestration, transactions etc, to define/create and execute telecommunication services. Moreover, the issue concerning the secure and controlled interaction between application components deployed in different domains (e.g., an enterprise domain and a network operator domain) has been considered in the case study as well.

The Telecommunication Case Study and its issues described in this deliverable have been presented at BIGG Bridging Global Computing with Grid [13] .

Annexes to this deliverable contain the detailed analyses of the issues presented below in a summarised way.

2 The Telecommunication Case Study

Telecommunication Services, i.e., the services that are provided by a telecommunication infrastructure managed by a public network operator, are evolving by considering several aspects of "convergence":

- convergence of media: the same service has to combine different types of communications, including voice, video, data streams (e.g., triple play);
- convergence of terminals: a service should be able to be activated by heterogeneous terminals (including PCs, mobiles, PDAs, TV SetTopBoxes), and telecommunication access networks (e.g. ADSL, UMTS, GSM/GPRS); moreover, the same service session could involve different types of terminals;
- combination of service features: a service may combine different telecommunication features, including multi-media communication, messaging, and content access;
- convergence of Telco and Internet/Web worlds: the distinction among the application contexts is disappearing: e.g., a telecommunication service could be configured, activated, and controlled by Web applications (e.g., chat applications enabling the activation of phone calls), or an IT application could integrate some Telco features (e.g., fleet management application integrating location features).

Moreover, the services should be "user-centric", i.e. their behaviour should be personalised according to the requirements of single end-users. The possibility of personalising should be uniform and cover all the features of the service. In particular, the end-users should be seen as single entities even across several different networks, terminals, communication media, and applications/services.

In general telecommunication services are created, executed and managed by a "Service Layer", which consists of a set of systems implementing the functions needed for the service delivery. The services are defined by a "business logic" that may use and combine a set of Telco features which provide some basic service capabilities. Examples of Telco features are:

- capabilities for controlling bearers/channels: call control (e.g., Service Switching Points in the switches), multimedia session control (e.g., Serving SCSF functions in IMS networks), Gateways for WAP connections, systems for handling SMS and USSD, etc.
- enablers for implementing VAS (Value-Added Services): servers for location, presence, messaging, terminal adaptation, etc.;
- data profiles: end-users profiles, service profiles, terminal profiles, end-users account, etc.

Services are either triggered by some events produced by Telco Features (e.g., call activation, the reception of a new message, the change of location of a terminal/end-user), or activated by some applications (e.g., deployed on an end-user terminal interacting through a specific protocol, such as HTTP, SIP, etc.). Currently, services interact with Telco Features and terminals through a wide set of protocols, each of which specialised to deal with specific capabilities/functions. Examples are: SIP, INAP, and CAP for session and call control, LIF for localisation, SIP for presence, USD for control of SMSs, MM7 for control of MMSs, etc.

Most of the current service layer is realised as a set of "vertical" platforms, named Silos, each of them specialised to provide services involving a specific Telco Feature and a specific network. In general, such platforms integrate in a single system the service execution environments with the Telco features and some supporting functions (e.g., payment, authentication, profiles).

In general, the vertical systems deployed in a service layer are loosely integrated:

- it is quite difficult to share a function across different platforms (i.e., to allow that a service deployed on a platform A can access a function implemented on a platform B);
- the same functions are duplicated on several platforms.

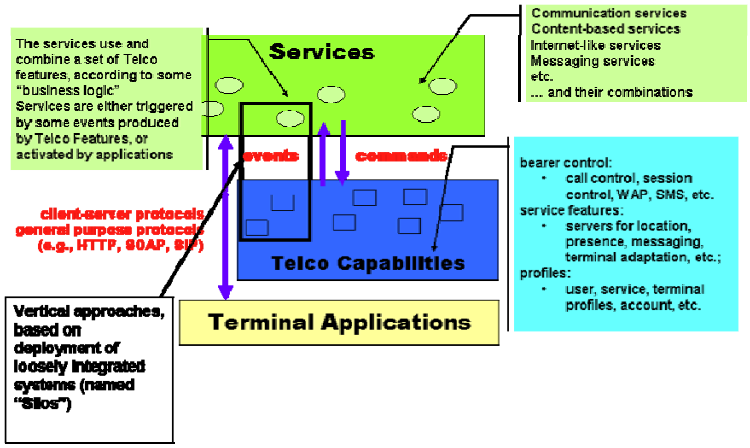

Figure 1 - Vertical Telco Architecture

Such an organisation of a service layer introduces several problems in dealing with the realisation of "converged" services.

In order to improve this situation, the service layer is evolving towards a "horizontal" approach based on:

- the integration among systems for service delivery which are deployed in the Operator infrastructure;
- sharing of and interoperability among functions, enablers and service capabilities.

At present there is not a shared definition of a horizontal Service Layer (SL) in the literature, but some common principles can be identified:

- all the functions in the SL should not be duplicated, and they must provide open interfaces to be accessible by either any service logic, or any other function in the SL;
- service enablers and bearer/channel control functions can be accessed through elements which provide an abstract/simplified view (e.g., in terms of used protocols and/or exposed functions);
- different types of service execution environments can coexist and cooperate for delivering services; each of them should be equipped with development environments/SCEs (i.e. Service Creation Environment);

- service logic (at server-side) must be deployed on a service execution environment; other SL and network functions could require some configuration to support the execution of the service logic;
- all data associated to different entities (customers, services, etc.) must be accessed through a unified view (even if the data may be stored in different repositories); moreover, the SL should allow the identification of customers in a unique way and independent of used terminals and access networks;
- a single set of functions and a uniform set of interfaces must be provided by the SL for the interaction with applications deployed in 3rd party domains (e.g. service providers, content providers, enterprises);
- SL functions must interact through a common communication infrastructure;
- interfaces towards OSSs and BSSs (i.e. different types of telco operator data systems) must offer a simplified view of SL elements, also abstracting the SL internal architecture.

A possible trend for the evolution of the SL according to a horizontal approach is the adoption of Web Services and Service Oriented Architecture (SOA) solutions. At the moment, there are already several initiatives that consider this possibility, such as the specification of Web Services for controlling Telecommunication services (e.g., Parlay X Web Services jointly specified by ETSI and 3GPP, the Mobile Web Services under specification by the OMA initiative, or Identity Web services specified by Liberty Alliance).

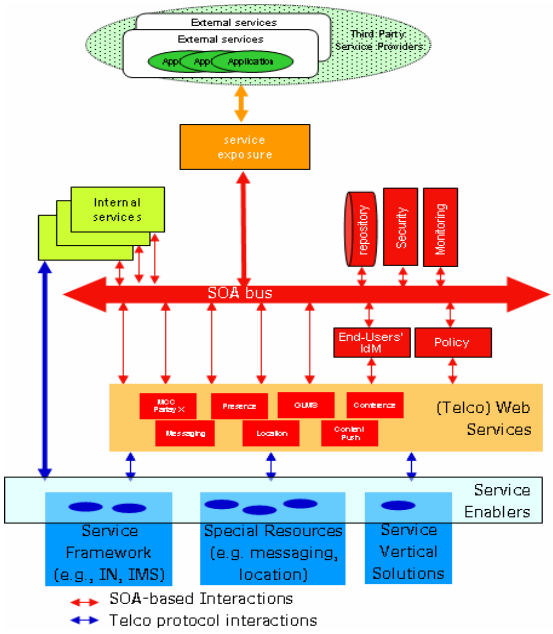

Figure 2: A possible model for a horizontal Service Layer

The SL model includes the following macro-functions:

- Service Execution: it contains functions for the deployment and the execution of the business logic of the services; typically the execution functions are carried out by one or more execution environments/application servers based on several technologies: J2EE and JAIN SLEE AS, BPEL AS;
- Service Exposure: it provides functions for secure and controlled interactions among applications deployed in 3rd party domains and the exposed services which are executed in the Service Execution;
- Telco Web Services (Telco Capability Abstraction/Adaptation): it implements a uniform set of APIs (e.g., based on an event/command interaction model) that provide an abstract view of the Network Control Functions and Service Enablers to other macro-functions of the SL, in particular to the Service Execution macro-function;
- Service Enablers: they are a set of functions that enable the deployment and the delivery of services based on features additional to the "basic" connectivity/session control; examples of Service Enablers are: communication enablers for Multi Media services (e.g., conference server), user interaction enablers for service-user/terminals interaction; messaging enablers to control messages exchange; info delivery enablers to control information and content delivery to users via Web, WAP, streaming; context enablers for context data processing, e.g., location Server, and presence.

The SL may be seen as a complex distributed system which must fulfill requirements coming from the telecommunication domain, such as the capability to interact with the network control functions and service

enablers, to provide suitable computational throughput, and support high-availability and fault-tolerance. In any case it must be able to interact with the Internet/IT domain in order to provide Telco-IT convergent services.

Some of the areas where the Web Services/SOA approach can be adopted for the evolution of the SL are:

- adoption of SOA-based principles and techniques (e.g., based on Web Services) to organise the internal structure of the SL, and to define the communication bus among the different services and macro-functions;
- evolution of composition mechanisms for creating and executing Telco Services by adopting solutions based on orchestration/choreography (those mechanisms allow the possibility to handle semantic and dynamic compositions of services);
- adoption of Web Services technologies to expose on the "Internet" Telco Capabilties/Features to 3rd party applications (e.g.. Service/Content Providers, Enterprises), and to assemble services provided by 3rd parties (e.g., to interact with payment services, to access contents);
- adoption of Web Services technologies to introduce a uniform interaction model among services delivered by the SL and terminal applications.

Figure 3 highlights the main SOA components in the above architecture.

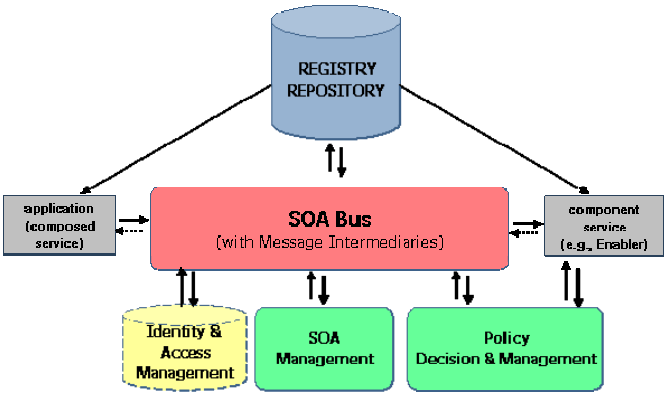

Figure 3 – SOA Components

- **Registry/Repository**: directory of the services, with all the relevant information useful for their usage and management
- **SOA Management** : management of the SOA infrastructure (e.g., monitoring of WS usage)
- **SOA Bus**: mediation functions for processing and controlling SOA messages, through Intermediaries (e.g., policy enforcement, security checks, rerouting, load-balancing, event notification)
- **Identity & Access Management**: systems to control the access to the services from applications and the involvement of end-users (e.g., privacy)
- **Policy Decision & Management**: handling of policies to control that the usage of enablers by applications fulfills the parameters defined at subscription time (SLA)

In order to follow this trend, several issues must be addressed and several open points must be solved. The suitability of the SOA approach to the requirements for the delivery of telecommunication services should be evaluated and possible improvements and extensions should be investigated.

2.1 Telecommunication Case Study issues

The Telecommunication Case Study issues described in this document aim to contribute to solve the following:

- Service Modelling (in relation to WP1 activity)
- Formal analysis of a protocol for identity management: end-user identity in a federated context, application vs. end-user identity (analysed in WP3);
- Protocol definition supported by formal analysis (studied in WP3);
- Synchronous and asynchronous service composition (WP5);
- Policies for exposing Web Services: dynamic negotiation / SLA (Service Level Agreement) and enforcement (WP5 analysis);
- Transactions for handling exceptions in service composition (part of WP5).

2.2 Relationship with other Sensoria WPs

This section aims to single out how the initial activities concerning the Telecommunication Case Study are related to the activities in the research WPs; in some cases the activities apply the initial results of the WPs to the issues of the case study, while in other cases they are motivated to identify possible requirements to be addressed by the WPs in order to solve open problems with respect to the state-of-art.

The SRML language for service modelling developed in WP1 was used to specify a telecommunication service, in order to verify the suitability of the approach proposed by SENSORIA in modelling telecommunication services, and to identify possible improvements.

The activity on the formal analysis of a protocol for identity management (to validate the security properties of the interactions, to identify possible threats) has been carried out by using the results of WP3 (T3.1).

An asynchronous protocol suitable for handling long-running computations and event notifications has been analysed: the methods and tools investigated in T3.3 and T3.4 have been applied to investigate the properties of a protocol for Web Services asynchronous invocation.

The Synchronous/Asynchronous Service Composition in order to develop or adopt a suitable language for composition of Telco components (e.g. Web Services), covering also the asynchronous communication aspects, has been faced with methods developed in WP5. Requirements for designing TLC services with asynchronous aspects have been identified by investigating the solutions that could be currently obtained by using BPEL: these requirements could be used, to outline the improvements/requirements on events handling, asynchronous service design and execution mechanism that should be addressed by the formalism to be investigated in WP5.

The activity on the specification of an advanced formalism (language), in order to define and (dynamically) negotiate and verify policies, has been formalised by using the results of WP5 (T5.1).

Transactional composition of Telco Web Services has been studied and analysed, applying the formalism investigated in WP5 and combining it with BPEL.

3 Service Modelling

The proposed service context, Call&Pay Taxi, concerns the retrieval and purchase of goods or services via a mobile terminal. The SENSORIA Reference Modelling Language (SRML) [14][15] will be applied for modelling the service scenario.

The services could be provided by a 3rd party Service Provider different from the mobile operator. The Service Provider could develop the service by assembling service components, which allow controlling either capabilities/functions provided by a mobile telecommunication network (e.g., control of voice calls, sending/receiving of SMSs/MMSs, localising terminals), or complementary resources and applications (e.g., to handle contents such as maps, banks accounts).

Specifically, the Call&Pay Taxi service provides a user with the possibility to call a taxi by sending an SMS to a specific SMS service number and to pay the taxi service by sending another SMS. The service will automatically debit the charging amount for the taxi service to the end-user's credit card, transferring the money amount to the taxi company.

From the point of view of the involved human actors (i.e., end-user, call center agent, taxi driver) the service behaves in the following way:

- The end-user sends an SMS to the SMS service number (e.g., 4777), which is associated to the "Call&Pay Taxi" service; in this way the end-user asks to call the taxi company of the town where he/she is currently located.
- The end-user receives an incoming call on its mobile phone, in order to be connected to the call center of the local taxi company.
- The end-user talks with the taxi company agent; the user and the agent perform the usual conversation in order to get the request of the user; the call center agent contacts the available taxi drivers and confirms to the user the selected taxi number/name and the expected waiting time.
- The end-user receives an SMS A, including the taxi number and a "call-code" to identify the ride.
- The taxi driver receives a similar SMS.

- After the taxi ride, the user replies to A with an SMS including the same information received and the amount to be paid, in order to authorise the payment on his/her preferred means of payment (which was stored in his/her user profile).
- In case of a successful transaction, the taxi driver and the end-user receive a confirmation of the payment with an SMS, reporting the taxi number, the "call-code" and the paid amount (and the end-user can leave the taxi); in case of failure, an SMS with such an indication is sent to them (and the end-user has to pay in the traditional way).

3.1 Modelling "Call and Pay Taxi" Service with SRML-P

The work presented in this section analyzes the orchestration and composition of telecommunication web services following an event-based approach. More specifically, the SENSORIA Reference Modelling Language (SRML) has been applied for modelling a service scenario that has been used by Telecom Italia as part of its research and development activities on Parlay X telecommunications web services – *Call and Pay Taxi through SMS*, or *Call&PayTaxi* for short. This is a service that provides an interface for booking a taxi using a mobile phone.

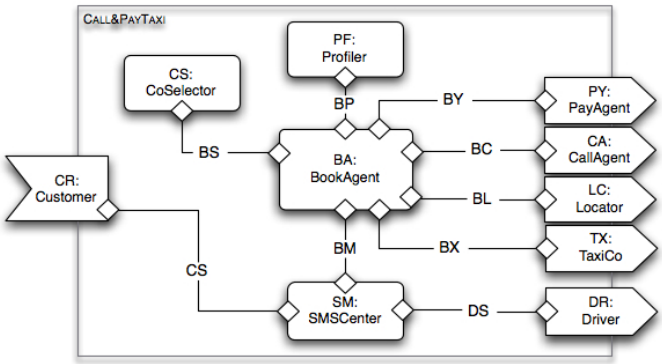

Figure 4: The SRML structure of Call&Pay Taxi

This exercise provided important information on the expressiveness and the suitability of SRML for modelling scenarios used in domain-specific contexts such as telecommunications. The first issue to arise during the modelling activity was the definition of the structure of the module for *Call&PayTaxi*: in order to define the structure one has to decide which entities of the scenario description are to be represented in the SRML module and, for those that are, whether they are internal components or external interfaces (EX-R for short). A component in SRML corresponds to a resource that is used internally in the sense that it is not visible to the clients of the service provided by the module. Such resources are tightly bound inside the implementations of the module and represent, as an ensemble, the orchestration of the module. An EX-R represents a dependency of the module on services that need to be discovered externally, i.e. beyond the internal components. For instance, it was decided to represent the service for sending/receiving SMSs, called the SMS centre, as a component, meaning that the service provider has its own SMS centre that is available via a fixed phone number, say 4777. This design decision triggered a refinement of the way SRML handles **module parameters** and to a better understanding of the role of **interaction parameters**.

- In order to support reuse and the definition of configurable services, the possibility of parameterised modules (and the specifications used in modules) was introduced; the module is configured at design time by instantiating the configuration parameters and defining their value. For instance, the parameter *serviceNum* is used to abstract from the concrete number that is used in any given service provided by an implementation of the *Call&PayTax* module. In the concrete example, this parameter is instantiated, at design time, with the number 4777; all sessions of this service are for calls that use the number 4777.
- Interaction parameters cope with the fact that every interaction event for a component can happen only once during the execution of a service instance. Sometimes, one wants to define operations that can be invoked an arbitrary number of times during the same session; therefore the use of interaction parameters for defining families of interactions is proposed. For instance, in the case of the SMS centre, the invocation of several SMS send/receive operations is supported by the definition of a family of interactions through a parameter – e.g. *sendSMS[k:int]*.

The *Call&PayTaxi* case study also provided a rich scenario for testing the modelling of interaction protocols in SRML. The interaction protocols involving the SMS centre manage the mismatch between parameters of corresponding interactions. The mismatch can concern the number of parameters (e.g., sending an SMS without text), or the type of parameters (e.g., creating textual messages from parameters and *vice versa*). These transformations are performed through operations *textify* and *parse* declared locally in the interaction protocols. The fact that these operations are declared locally means that they need to be implemented whenever the interaction protocol is instantiated with specific data sorts.

The *Call&PayTaxi* scenario is being used as a test bed for the development of SRML. As the current understanding of what is required from SRML as a service modelling language grows, the syntax of the language evolves and its semantics becomes closer to the intended usage. Because of this, the model proposed here is in no way supposed to be final. These models will be refined as SRML develops. For instance, a number of issues that require the extension SRML-F for configuration management have been omitted.

The annexes contain the SRML-P "Call and Pay Taxi" Service Model with the detailed specification. The refinement of the language is carried on in WP1 of SENSORIA.

4 Formal analysis of a protocol for identity management

The main objective of this activity is to provide a formal analysis of a protocol aiming at transferring identity and authentication information handled by the systems in an operator's network infrastructure to the applications implementing services provided by 3rd party service providers. The formal analysis has to verify the correctness of the protocol and investigate its security properties. The results of the analysis will be considered in order to identify possible flaws or weaknesses of the protocol and to suggest improvements.

The analysed protocol was defined by Telecom Italia in some activities external to SENSORIA [1]. It is a specialisation of the Liberty Alliance framework [2] to the context of a telecommunication operator, in order to avoid requesting explicit credential to end-users, by exploiting the network infrastructure of a network operator. In this way, a network operator may play the role of Identity Provider, with respect to Service Providers belonging to a Circle of Trust.

In particular, the protocol handles identity and authentication information of end-users accessing Internet applications through different access networks (e.g., ADSL and GPRS/UMTS). It supports the translation of the authentication/identity information provided by the access network to the application level and adds it to the HTTP requests. In this way the end-users can access the services without explicitly providing any credentials and the applications can rely on the authentication/identity information introduced by the protocol.

The protocol relies on the association (federation) of the identities of end-users of a telecommunication network with the accounts of the Service Providers' applications, represented by a Token (or Opaque-id). Thanks to such an association mechanism, the identity information transported by the protocol keeps the anonymity to the end-users, but allows the applications to condition the access only to the authorised end-users, and to personalise the services according to the account associated to the transported identity information.

[1] provides a high-level description of the protocol and of the involved systems, for the HTTP case[1]. The main component of the solution is the Token Injector. It should be deployed in a secure network (i.e., a network that provides a high-level degree of security on the communication between the interconnected systems/terminals), such as the ones implementing access to telecommunication infrastructures (e.g., GPRS, or ADSL); it must be able to intercept all the data traffic originated by an end-user terminal connected to the secure network and terminating into an application server through an insecure network. It must be able to introduce in the intercepted messages some identity information of the end-user. The format of such a token must be compatible with the application protocol (e.g., HTTP) used in the interaction.

When a user performs the request to access a specific service, the Token Injector, deployed in the network operator domain, intercepts the HTTP GET (or POST) request and, depending on the information stored on the Operator's repositories (user profile, service profile, etc.), inserts an authentication "token" (i.e. a string of characters) into the original HTTP packet and forwards it to the final destination, for instance to a Service Provider connected to the Internet. When the application server receives the new access request, it looks for an authentication token. If such a token is present, and it is a valid one (e.g., issued by a trusted authority, well formed and so on), the user is immediately recognised and accepted without any explicit request for credentials because of the token.

[1] The described protocol may be adapted to several protocols, including SOAP and SIP.

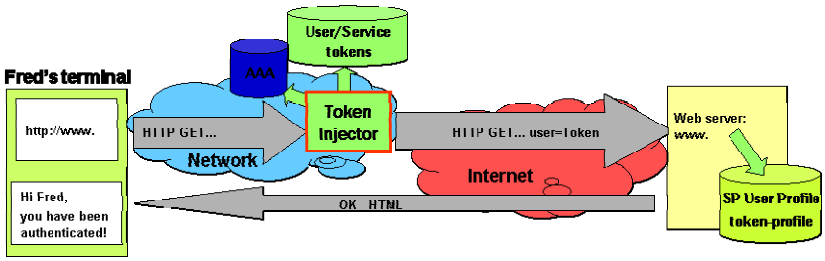

Figure 5: Identity Transport protocol

In more details, when a terminal performs an HTTP request to access a given URI, the Token Injector intercepts it, and, if the URI belongs to one of the affiliated services, it performs the following steps:

1. get the IP address of the sender of the HTTP request packet; access the AAA server of the access network in order to get the information of the end-user identity (EU-ID) in the context of the access network (e.g., IMSI or MSISDN in case of mobile network) associated to the IP address; in general, the IP address has been assigned to the terminal in a secure manner by the access network and it is very unlikely for an unauthorised user to spoof other IP addresses;
2. access the repository storing the associations between <EU-ID, URI, Token> in order to get the Token corresponding to the requesting end-user; if the association is found, the Token Injector modifies the HTTP request in order to add the token (in a way that can be processed by the application), and forwards the modified request to the requested application server;
3. if such an association does not exist, the Token Injector forwards the original request without any modification to the application identified by the required URI; in this case, the application could return a registration page for asking the end-user to subscribe to the service.

In case the requested URI is the reference to a subscription/registration page, the Token Injector executes a variant of step 2: it creates a new tuple <EU-ID, URI, Token> with the creation of a new Token. If there is already such an association, the Token Injector forwards the existing Token to the application which is in charge to decide how to deal with this particular exception.

The token can be formatted according to specific standards, such as SAML [3] and it can carry additional information, such as a description of how the end-user has been authenticated, or some attributes associated to the end-user (e.g., preferred language).

Moreover, the protocol is able to exchange information between "federated" Token Injectors, each of them associated to a specific network access infrastructure (e.g., wireline, mobile), in order for the service providers to receive the same identity information, independently of the access network used by the end-users.

Finally, the protocol is able to deal with the identification of end-users by means of temporary tokens, in order to avoid that an application is able to make correlations among requests performed by the same end-user.

The protocol analysis will consider the following main features:

- subscription to a service, with the transparent federation between the new account created by the service provider and the identity of the end-user in the telecommunication domain;
- access to a service, with the automatic introduction of identity and authentication information needed by the service provider, by considering support to privacy and anonymous accesses;
- interactions between federated identity providers, in order to guarantee that the service providers receive the same identity information, independently of the access network used by the end-users.

4.1 Description of the formal analysis and of the approach

Formal methods and tools are popular means for the analysis of security aspects of computer network protocols. First the protocol under scrutiny is described in a formal language, which often results in a more precise definition of its functioning. Subsequently, the security aspects to be analysed are specified in a logic. Finally, to decide whether or not certain security properties are fulfilled by the protocol, an automatic tool is used to analyze the protocol. The outcome either proves the protocol correct w.r.t. the security aspects considered or shows how it falls to one or more attacks. The literature contains many examples.

In [10] three user scenarios of the above identity federation protocol for services in convergent networks are formally specified and some of its security aspects (like the possibility of a man-in-the-middle attack) are analysed. To do so, Crypto-CCS, a CCS-like process algebra with cryptographic primitives, was used in combination with the Partial Model Checking Security Analyzer PaMoChSA developed by the Security group of IIT-CNR (see http://www.iit.cnr.it/staff/fabio.martinelli/pamochsa.htm).

The analysis approach of [12] was adopted. This approach is based on the observation that a protocol under analysis can be described as an open system: a system in which some component has an unspecified behaviour (not fixed in advance). Subsequently one assumes that, regardless of the unspecified behaviour, the system works properly (i.e. satisfies a certain property). In case of the network protocol of the previous section, one can imagine the presence of a hostile adversary trying to interfere with the normal execution of the protocol, in order to achieve some kind of advantage w.r.t. the honest participants. Such an adversary is added to the specification of the network protocol as a component with a behaviour that is defined only implicitly by the semantics of the specification language.

The adversary is assumed to act in a so-called Dolev-Yao fashion by using a set of message manipulating rules that model cryptographic functions like encryption and decryption. Encryption is opaque, i.e. a message encrypted with the public key of one of the participants cannot be decrypted by anyone but the person who knows the corresponding private key (unless the decryption key is compromised of course). As is common in this branch of computer security, a black-box view of cryptography is adopted, by assuming all cryptographic primitives involved in the network protocol to be perfect. Like the honest participants, the adversary is able to send and receive messages to other participants. However, it can also intercept and forge messages and, to a certain degree, derive new messages from the set of messages that it has come to know. This set consists of all the messages the adversary knows from the beginning (its initial knowledge) united with the messages it can derive from the ones intercepted during a run of the protocol. To analyze whether a system works properly, at a certain point in the run the adversary's knowledge is checked against a security property.

If the intruder has come to know information it was not supposed to know, then the analysis has thus revealed an attack w.r.t. that particular property, i.e. a sequence of actions performed by the adversary that invalidates the property. The first scenario formalised in [10] is that of federated registration:

$$\begin{array}{lllll} c_0 & U \mapsto IdP & : & r \\ c_1 & IdP \mapsto SP & : & \{r, SAML\,assertion\}_{K_{IdP}^{-1}} \\ c_2 & SP \mapsto U & : & \{ok/ko\}_{K_{SP}^{-1}} \end{array}$$

The entities involved are a user U, a service provider SP and a network/identity provider IdP: U asks for a registration to SP and for a federation between SP and IdP (sharing a secret key) in the sense that - once federated - IdP and SP may provide U access without directly asking for any credentials, but by simply relying on the information given by IdP. Federated registration consists of three main phases: the authentication phase in which IdP authenticates U, the token generation and the assembling of a so-called SAML response to be sent from IdP to SP, appropriately signed by the private key of IdP. For modelling purposes only, the SAML response is identified with a SAML assertion and properly encrypted: it contains a field Subj with the token id_U univocally identifying U, a field AuthStat with an authentication statement asserting that U was authenticated and the mechanism under which she/he was authenticated and, finally, a field AttrStat = < attr_list, n_U^{IdP} > with a list of attributes of U related to her/his service accesses (e.g. country preferences if the service is a travel agency) and a nonce to avoid replay attacks. Note that the encrypted SAML assertion conforms to the SAML standard (see http://www.oasis-open.org/specs/), and that communications are appropriately signed.

Next a specification of this federated registration scenario is given in Crypto-CCS. This specification is more expressive than the one above, because all operations and security checks on the various messages are explicitly modelled. The reader is referred to [10],[10] for the details of all three scenarios.

Finally, as an initial step towards a full-blown security analysis of the protocol, the vulnerability of the first two scenarios w.r.t. a man-in-the-middle attack was checked. A man-in-the-middle attack is an adversary's attempt to intercept and modify messages between two trusted participants, in such a way that neither participant is able to find out that their communication channel has been compromised. Model checking is used to perform the analysis. This is an automatic technique to verify whether or not a system design satisfies its specifications and certain desired properties. Such a verification is moreover exhaustive, i.e. all possible input combinations and states are taken into account. The level of completeness of a verification of course depends on the range of properties that are verified. Compared to testing, model checking generally needs to be performed on an abstract system (specification) to avoid state-space explosions. However, more problems are usually found by model checking the full behaviour of a scaled-down system than by testing some behaviour of the full system.

In [10], e.g., the specification of the insecure channel between IdP and SP was checked against a man-in-the-middle attack, i.e. an adversary trying to intercept a conversation between IdP and SP. This boils down to verifying the following property: whenever SP concludes the protocol apparently with IdP, it was indeed IdP executing the protocol. To check this, the model checker PaMoChSA requires the following input:

- a file with the protocol specification in Crypto-CCS;
- a logic formula expressing the property to be verified;
- the adversary's initial knowledge.

An adversary X was taken in consideration and its initial knowledge was set to the set of public messages that it knows at the start of the protocol: the public keys of IdP and SP and its own public and private key. Consequently, the result of the performed analysis was as follows: NO ATTACK FOUND. It took the tool less than a second to conclude this. Hence the protocol is correct w.r.t. the analysed security property, i.e. it does not fall to a man-in-the-middle attack.

The result of the presented security analysis strengthens one's confidence in the formal specifications of the three user scenarios presented in [10]. In particular, it leads one to believe that the digital signatures, encryption and nonces were inserted into the network protocol in a correct way. This is a clear advantage of the use of formal methods in the design phase of a protocol: it allows one to eventually arrive at a well-defined protocol that is guaranteed to satisfy certain desirable properties. In the annexes, the referred papers report all the details.

5 Protocol definition supported by formal analysis

The main objective of this activity is to define a variant of SOAP supporting asynchronous communications, driven step-by-step by the results of a formal analysis. This activity is, thus, different from the one described in Section 4, where the formal analysis is performed on a protocol already specified to verify its correctness.

The domain of the research is the definition of a SOAP-based protocol which supports asynchronous interactions, i.e., interactions different from the usual synchronous "request-response" interactions supported by the available SOAP implementations based on HTTP. Such kinds of interactions are quite relevant in the delivery of telecommunication services, as:

- service logic are triggered/activated by events produced, in an asynchronous way, by the network/special resources, or must react to such events during the execution of a service instance;
- requests produced by a service logic to a network/special resource may results in long-running computations (e.g., set-up of a call) which might also require the involvement of end-users;
- some service logic components may not be reachable (e.g., the ones deployed on mobile terminals), e.g., due to the temporary absence of communication.

The objective is to define a protocol, named aSOAP, which is able to address most of these situations. The following are the requirements to be considered:

- Backward compatibility:
  - aSOAP must be compatible with SOAP v1.2 on HTTP [4];
  - aSOAP must have limited impact on the clients: i.e., the clients that do not need to support asynchronous interactions must be SOAP clients, working according to a request-response mode; clients that need such a support should introduce limited variants w.r.t. normal SOAP requests;
- Reachability
  - aSOAP must be able to deal with unreacheability of the servers (e.g., due to lack of connectivity)
  - aSOAP must be able to deal with the case in which a server cannot return a response (either provisional or final) due to the lack of connectivity
  - aSOAP must be able to deal with the case in which a response (either provisional or final) cannot be returned to a client due to the lack of connectivity
- Message Exchange Patterns
  - aSOAP must be able to deal with requests that require the servers to perform some long-duration processing (greater than HTTP time-out) before producing any results
  - aSOAP must be able to deal with requests with multiple responses

The protocol may rely on some intermediary nodes, which are assumed to be highly available. During the process for defining aSOAP some of the requirements could be relaxed. Paper [7] describes a possible functional architecture supporting aSOAP, where aSOAP proxies are SOAP intermediary nodes.

Figure 6: aSOAP architecture model

The formal analysis of a preliminary definition of the protocol highlighted several issues and flaws that suggested to completely reviewing it.

In a second phase quantitative analysis will address issues related to the performance of the protocol.

5.1 Description of initial activities and results

In [7], a first step in the development of aSOAP is presented. This step consists of the use of formal methods to analyse an initial formalisation. The formal model of aSOAP is specified as a set of communicating UML state machines. UML is used as particular formal method since it has become the de facto industrial standard for modelling and documenting software systems. The UML semantics associates a state machine to each object in a system design, while the system's behaviour is defined by the possible evolutions of the resulting set of state machines which may communicate by exchanging signals. All these possible system evolutions are formally represented as a so-called doubly-labelled transition system, in which the states represent the various system configurations and the edges represent the possible evolutions of a system configuration. Subsequently, several behavioural properties of the aSOAP model are represented in the action- and state-based temporal logic μ-UCTL and checked with the on-the-fly model checker UMC that allows one to check UML state machines.

UMC is a model checker that creates and traverses the state space on the fly. The advantage of on-the-fly model checking is that often only a fragment of the full state space needs to be generated and analysed to obtain a satisfying result. The development of UMC is still in progress and a prototypical version is being used internally at ISTI-CNR for academic and experimental purposes. So far, the focus of the development has been on the design of the kind of qualitative features one would desire for such a tool, experimenting with various logics, system modelling languages, and user interfaces. The quantitative aspects of such a tool, e.g. concerning optimisations aimed at limiting state-space explosions or the complexity of the evaluation algorithms (to their known minimal limits), have not yet been taken into consideration. There has not yet been an official public release of the tool, even if the current prototype can be experimented via a web interface at the address: http://fmt.isti.cnr.it/umc/.

The behavioural properties that UMC can check need to be expressed in the aforementioned action- and state-based temporal logic μ-UCTL, which includes both the branching-time action-based logic ACTL as well as the branching-time state-based logic CTL. Starting from the basic μ -UCTL operators, one can derive the standard μ-calculus operators and one can of course also derive the standard CTL/ACTL-like temporal operators such as EF ("possibly"), AF ("eventually"), AG ("always"), and the various Until operators, in the usual way.

Before discussing several aspects of the formal specification of aSOAP, first the assumptions that are part of the design of aSOAP are recalled:

- The Proxy is always reachable by both the Client and the Server whenever they have an active connection;
- If the Client is willing to accept an asynchronous response to its SOAP Invocation, then it inserts in the SOAP header the URL of the SOAP listener where it wants to receive the response;
- The URL in the header of an asynchronous SOAP Invocation is the address of a generic SOAP listener and the application level is equipped with a mechanism for receiving SOAP messages at this URL;
- Upon receiving an asynchronous SOAP Invocation from the Client, the Proxy generates a Request Identifier REQ-ID that uniquely identifies the Client's SOAP Invocation in further communications.

Regarding the first assumption it is important to note that the considered scenario is such that both the Client and the Proxy reside in the Service Layer of an operator. The application that may invoke web services (i.e. the Client) is thus connected with a high level of stability. In particular, services are not invoked from a mobile terminal with unstable connectivity. In the considered scenario it is the Server that might be a mobile terminal. Moreover, a mobile scenario is only one of the scenarios considered, in the sense that aSOAP might be used also to handle requests that activate long-running computations.

During several sessions between ISTI-CNR and Telecom Italia, the design of aSOAP and its formalisation were discussed and developed in detail. In order to facilitate the discussions about the behaviour of the various use-case scenarios of aSOAP, it was decided to consider a separate message sequence chart for each such a

scenario. Finally, all these scenarios were translated into an operational model, in which the following concrete modelling choices were adopted:

- All SOAP invocations are asynchronous, i.e. the synchronous SOAP invocations that only serve to guarantee backward compatibility with SOAP v1.2 are abstracted from;
- The URL in the header of a SOAP message is identified with the Client, i.e. each Client is seen as just a listener of asynchronous SOAP invocations;
- A system model is constituted by a Server (and its subthreads), a Proxy (and its subthreads) and a fixed (configurable) number of Clients;
- The Proxy and the Server may activate at most a fixed (configurable) number of parallel subthreads;
- With the Client or the Server unreachable, the Proxy attempts to contact them up to a configurable number of times;
- The Client issues a single SOAP invocation and then terminates. (In the future, Clients that perform a loop of SOAP invocations or that issue several SOAP invocations before waiting for the deferred SOAP results will be considered.)

The full specification of aSOAP can be found online at the address:

http://fmt.isti.cnr.it/umc/examples/0-ASOAPPP.umc

These abstractions are sufficient to allow model checking an initial set of properties with UMC. To begin with, the state-space complexity of the aSOAP specification was studied. This revealed that the number of states increases exponentially with the number of Clients, which is of course due to the explosion of possible interleavings that is inherent to system models with more than one Client. As a result, already many system models with three Clients have state spaces that cannot possibly be traversed in full - let alone reasoned about without using automatic analysis tools. Since UMC is an on-the-fly model checker, however, certain properties can be checked for such system models and even for much larger system models. Note also that the maximum number of parallel subthreads that the Proxy and the Server can activate influences the total number of states of a system model only in system models with more than one Client.

Next some behavioural properties expressed in μ-UCTL were verified in a system model with 2 Clients, with a Proxy that may try at most 2 times to contact the Client or the Server in case of their unreachability and that may activate at most 2 parallel subthreads and with a Server that may activate at most 2 parallel subthreads. First, Property 1:

"All system executions eventually reach a configuration in which all Clients are in status 'Done'"

was checked by verifying the formula AF ((C1.status=Done) and (C2.status=Done)).

The formula turned out to be false, i.e. Property 1 does not hold. The reason is the fact that the Server's response need not reach the Client: A possible system execution (which can also occur in a minimal system with just one Client and no parallelism) is such that the Client's SOAP invocation is being deferred by the Server, but its subsequent final SOAP result never reaches the Client because the Client becomes unreachable for a sufficiently long time for the Proxy to cancel the SOAP invocation.

Interesting enough, Property 1 can easily be checked also for system models with tens of Clients (in case of a system model with five Clients, e.g. only 117 states need to be explored in depth-first mode). This is because UMC is an on-the-fly model checker that, as said before, only generates and analyses the fragment of the full state space that is needed to obtain the result.

The question is whether Property 1 does hold in a setting in which the Client and the Server are always reachable. Second, Property 2:

"For all execution paths without communication failures the system will eventually reach a configuration in which all Clients are in status 'Done'"

was checked. The formalisation uses a minimal fixed point structure and is as follows:

```
min Z :  ((C1.status=Done) and (C2.status=Done))
           or (PT1.result=Client_Unreachable)  or (PT1.result=Server_Unreachable)
           or (PT2.result=Client_Unreachable)  or (PT2.result=Server_Unreachable)
           or (not FINAL and [true] Z)
```

This formula is true. Hence Property 2 does hold. To obtain this result, UMC analysed 34735 states. While Property 2 can still be checked for system models with 3 Clients by exploring up to 96928 states, this is no longer the case for system models with more than 3 Clients.

Finally, the simplest system is considered, i.e. with just one Client, with a Proxy that tries to contact the Client or the Server only once and that may not activate any parallel subthread and with a Server that may neither activate any parallel subthread. Property 3:

"If Client receives a SOAP_Result(ReqId) operation call then it received a [Soap_Deferred,ReqId] response to its previous SOAP_Invocation"

was checked by verifying the formula AG [C1.SOAP_Result(*,ReqId)] (C1.result=[Soap_Deferred,ReqId]).

While this formula should obviously be false, in the considered model it is actually true, i.e. Property 3 does not hold. The reason is that the Proxy may find the Client unreachable, and thus be unable to notify the Client of the deferred Server response and of the REQ-ID that it has generated for its request. This of course does not prevent the request to proceed its usual course, until eventually the deferred result is produced by the Server. However, in this particular scenario the REQ-ID associated to this result will mean nothing to the Client. The gravity of this particularity and whether there is a way to avoid it is currently under study.

In the annexes, the paper presented at ECOWS'06 reports all the details of the analyses.

6 Synchronous and asynchronous service composition

While Section 5 addressed the problem of asynchronous interactions from the point of view of the communication protocols, this section addresses the problem from the point of view of service logic interacting with resources characterised by asynchronous interactions (e.g., notification of events), as described by [9].

The objective of the activity is to investigate orchestration/composition formalisms for defining telecommunication service logic interacting with component services which provide an abstract view of the telecommunication capabilities.

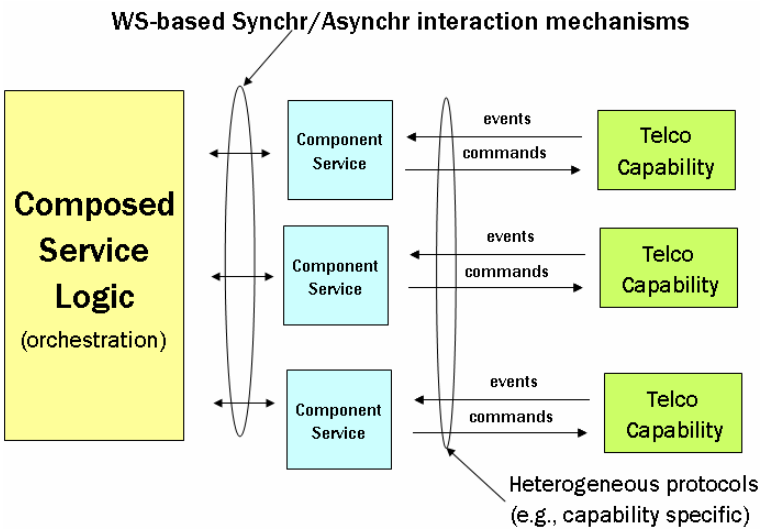

Figure 7: Synchronous/Asynchronous composition

From the "logical" point of view a component service may provide the following interaction primitives: Request-Response, Request without Response (i.e., one-way message), Notification, Solicit-Response. These primitives are implemented by a set of Web Services (e.g., the Parlay X ones specified by 3GPP/ETSI [5]), that may be used by the composed service logic to interact with the component services.

Document [9] identified some interaction patterns between composite and component services which are relevant in the context of the telecommunication services:

- Request-Response: it is the normal request-response interaction supported by SOAP/Web Services; it may be applied when the composed service may return the "final" result of the request processing, or when the composed service is just interested in an acknowledgment that the processing was started;
- Request-Response with continuation: it represents the case that the request starts a long-running processing, that could pass through different intermediate states; the composed service is interested to be informed on the changes in the processing status and/or to modify the processing; in this case the

initial Request-Response could be followed by a set of message exchanges between the composed service and the component service;

- Notification: the composed service subscribes to be notified of an event produced by a component service; the composed service has to define the logic that must be executed in order to process a notified event;
- Solicit-Response: the composed service subscribes to be interested in handling some events; the result of the processing is returned to the component service which continues its processing according to the received results; this interaction pattern may be used to allow a component service to perform an "Assist" request to a composed service to request some information useful for the composed service;
- Solicit-started transactions: it is an extension of the Solicit-Response interaction pattern; when a composed service receives a Solicit to handle an event, it may start a "transaction" for its processing which may involve several interactions with the service component; a composed service could process in parallel several transactions, related to the handling of different solicits;
- Request-started transactions: it is an extension of the "Request-Response with continuation" interaction pattern, where more complex interactions may be performed between the composed and the component services.

In order to better identify which are the critical points to address the identified interaction patterns by a formalism for composed services, it was decided to start investigating how they can be implemented by using BPEL constructs (e.g., correlation sets, pick).

In order to demonstrate the approach, a service scenario was defined based on the control of a multiconference, where the service logic has to interact (according to some asynchronous interaction patterns) with the resources controlling the multiconference. Figure 8 shows the reference architecture of the basic service scenario. A detailed description of this scenario, as well as of an extended scenario, are given in [6].

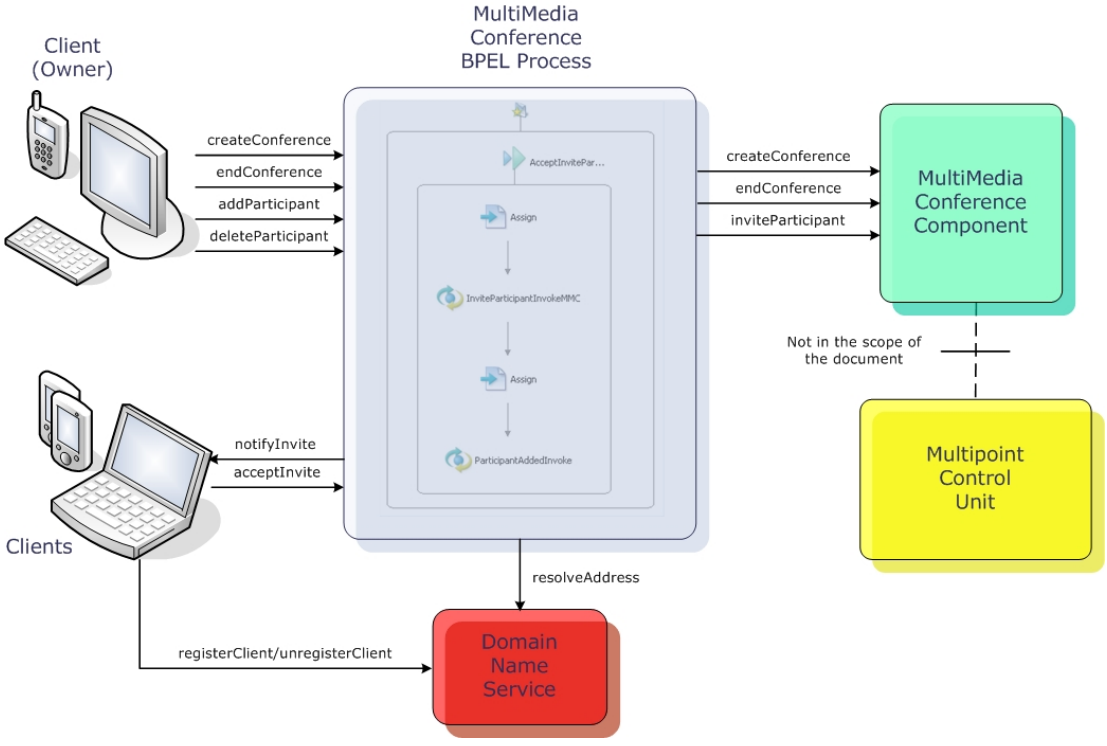

Figure 8: Reference architecture of the MultiMedia Conference (MMC) service scenario

In this basic scenario different actors are involved:

- **Client owner:** identifies the user who manages the multimedia conference; the Client owner can create and terminate a conference, and he can also invite or delete new participant users. It is assumed that only Client Owner can invite a new participant to the conference.
- **Clients:** identifies the new participants to a multimedia conference created by the Client Owner. A new participant is invited to a conference and is notified of this invite. In order to receive the invite notifications, a Client must previously register to a DNS service providing its notification reference

address (e.g. a URL of a notification Web Service). The client must accept the invite to connect to the multimedia conference.

- **BPEL MCC Process:** the BPEL process managing the multimedia conference. It receives Client requests and manages the multimedia conference service logic by Web Service orchestration.
- **DNS:** the Domain Name Server receives registrations by the clients and provides information about clients to the BPEL MCC process when the process needs to send an invite notification to a new client.
- **MCC Web Service:** is the interface module between the MMC BPEL process and the resource adaptors. The MCC WS provides the same interface to the BPEL process independently from the resource adaptors used in the implemented solution. The architecture used is the same adopted and defined in the Parlay X approach.
- **MCU (Multi Conference Unit):** is the module implementing the network functionalities. These interfaces are depending on the network solution adopted in the implementation and are out of the scope of this document.

All actors involved are assumed to provide Web Service interfaces for their interactions.

6.1 Preliminary results of the BPEL analysis to implement the pattern

The analysis uses the fact that the various patterns depicted above rely on a restricted set of requirements, combined in different ways. Whether BPEL is capable to support such patterns therefore depends on whether such requirements can be implemented using such a language. In particular, all or some of the patterns require:

1. the ability to handle asynchronous long-running interactions;
2. the ability to handle and synchronise parallel threads of computation;
3. the ability to refer to communication contexts, in order to correlate different communications pertaining to a same logical flow of data, and to dispatch them adequately to associated computation threads;
4. the ability to suspend activities on messages, and to resume them upon timeouts;
5. the ability to handle the association of sets of services to sets of events, which is required in subscribe-notification mechanisms.

For instance, the request-response pattern only requires (1) and the request-response with continuation pattern only explicitly requires (1) and (3).

All of these requirements, except for (5), are directly supported by BPEL constructs, and hencefore their combination can be easily realised as BPEL protocols. In particular:

- long-running asynchronous transactions are supported by BPEL's (non-blocking) *invoke* and (blocking) *receive* constructs. (Note that a synchronous variant is also available, through the *invoke/receive/reply* activities: whether an invoke is synchronous or not appears in the WSDL associated to the BPEL code).
- parallel threads are supported by either generating process instances upon message receiving (via the *createInstance* keyword), or by explicit forking of threads via the *flow* activity. They can be synchronised by BPEL *link* activities.
- The creation and correlation of communication contexts is handled by BPEL's *correlation* construct. Correlations build upon the definition of *properties* typically associated, via an aliasing, to some part of a message type. In this way, messages convey 'keywords' that are used to dispatch them to the properly correlated process instance. Once generated upon sending of a message, both sender and receiver can use the correlation set associated to the message to restrict their communication context to exactly the right partner instance.
- Timeout-based suspension of activities is easily handled by combining the *pick* and *onAlarm* BPEL constructs: the former allows suspending over several events by means of the *onMessage* BPEL construct; the latter specifies a timeout (in terms of relative or absolute time), and can be used as one of the suspending activities.

For what concerns the handling of associations of sets of services to sets of events, this would require a further "*associator*" construct in BPEL, which by now is not supported, but can be explicitly programmed in several ways; e.g., in terms of queues or associative lists. Note that such programming is particularly easy in a more restricted setting where no more than one service is associated to one event; in this case, it is possible to use correlation on event IDs over parallel threads to pursue a one-to-one correspondence.

Therefore, BPEL allows for a direct realisation of most of the patterns shown in [9], and of all of the remaining patterns once the "associator" construct is appropriately programmed. In particular:

- request-response can be realised simply by means of synchronous BPEL invoke/reply;
- request (with or without) response with continuation can be realised by means of synchronous and asynchronous BPEL invoke/reply, also using BPEL correlation;
- request-started transaction can be realised by means of synchronous and asynchronous BPEL invoke/reply constructs, using BPEL correlation and creating parallel threads to handle different execution contexts; the same holds for the W3C definition of a solicit-response pattern;
- the BPEL realisation of solicit-based interaction as defined in [9], as well as of the notification pattern, also require correlated synchronous and asynchronous communication and parallel threads, which can all be accommodated in BPEL. However, they also make use of the "associator" element, which as stated above is not supported natively in BPEL, but can be explicitly realised using BPEL constructs.

In an experimental way, the results of this analysis were verified by designing and testing prototypical BPEL implementations of the patterns above.

The annexes provide a description of interaction patterns, details on the multiconference service and examples of BPEL processes.

7 Policies for exposing Web Services: negotiation and enforcement

One of the critical points in SOA for the Telco Service Layer is the handling of policies to control that the applications use the enablers according to the subscribed parameters (SLA, i.e. Service Level Agreement).

The scope of this activity is to define languages to define Policies for the description of conditions to control the usage of Service Capabilities implemented in a Telecommunication Network and exposed to 3rd party service providers, external to the Network Operator domain. Such policies would contribute to define the SLAs between the Network Operator and the Service Providers.

Moreover, the activity aims at identifying mechanism to enforce such policies to control the usage of the Service Capabilities. Finally, it considers the problems concerning the negotiation of such policies, during the phases of subscription of a Service Capability and definition of the corresponding SLA.

Policy Decision & Management should support general-purpose policies (enforced by the SOA Bus by message intermediaries); component service specific policies (enforced by the component implementation); possible involvement of external decision points (e.g. end-user account management).

The aspects that need to be developed are: identification and specification of policies; support to deal with the dynamic negotiation of SLA; support to multiple points of policy enforcements in federated contexts.

Policies in general are a way to express:

- a control in order to guarantee that the service usage, i.e. on invocations of interfaces and operations of the Web Service, is compliant with the SLA agreed at subscription time;
- condition for the control, configuration and protection of telecommunication capabilities and end-users involved in a service execution.

Different types of policies can be identified, more specifically the policies can be categorised as:

- policies associated to the subscription of services (example Telco/IT adapter, application/Protocol): policies on Authentication (policy on the authentication mechanisms, e.g. IP address, username/password, …); policy on Authorisation (e.g. access on subscription; free access);
- policies related to the use of services, e.g.:
  - on time range (e.g. when a certain operation can be invoked); on invocation frequencies (e.g. how many times an operation can be invoked in a certain range);
  - on end-users' network addresses (e.g. are the addresses valid?; is the network operator involved a valid one?); on hiding of end-users' network addresses (it is indicated if the address can be explicitly passed or has to be hidden, e.g., by mapping addresses and aliases); on parameters' value (e.g. restriction on certain values);
  - on the session validity (e.g. it should be verified the link session/application, validity of the session); on the state of a session (e.g.. number of participants to a conference);
- policies associated to the user involved in the execution of the services (example: rules on the end user data of an application/Service Enabler, privacy rules).

The activity will consider, in particular, issues concerning the definition and enforcement of policies involving the correlation of multiple messages, and session status.

In addition, there are several policies defining constraints on the involvement of end-users in the service capability execution (e.g. constraints on sending SMSs, calling them, localising, etc.); the activity will consider issues on how to address and resolve possible conflicts among policies of the same or of multiple end-users.

Finally, the activity will consider how to define the policies, as a result of a negotiation between a request of a service provider and an offer of a service capability.

In order to illustrate the different policies listed a service scenario is introduced, i.e. Call By SMS and the different policies are described on the different flows that compose the service.

The "CallBySMS" service allows a mobile phone user to activate a voice call by sending an SMS message to a specific service number. The SMS message must contain a nickname of the person the user wishes to call. The service is able to automatically find the number associated with the nickname and to set up a 3rd party call between the user and the callee. In order to keep privacy, the service does not know actual phone numbers, but only opaque-id's representing users.

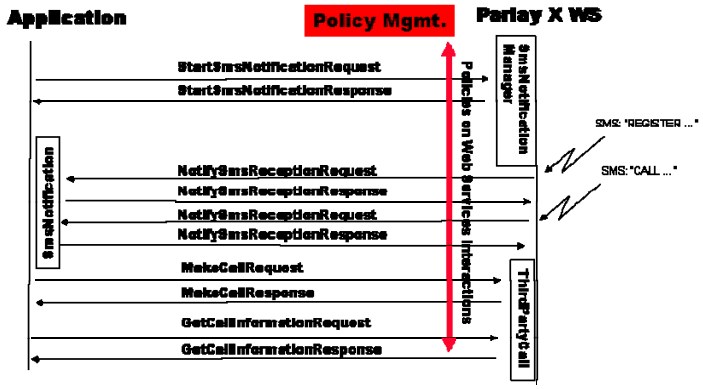

Figure 9: Call By SMS interaction flow

Figure 9 describes a scenario in which John wishes to call Mary: he knows that Mary's nickname is "sunshine":

- Mary sends an SMS message "REGISTER sunshine" to a given service number (e.g. 11111);
- The service associates "sunshine" to the phone number/opaque-id of Mary;
- John writes the SMS "call sunshine" and sends the message to a given service number (e.g. 11111);
- The service retrieves the phone number/opaque-id associated to "sunshine" and sets up a call;
- John's phone rings; John answers and gets the ringing tone;
- Mary's phone rings; Mary answers;
- John and Mary are connected.

In order to implement the application, the enterprise may subscribe two Telco Web Services, for instance, the Parlay X Web Services:

- SmSNotification, to receive the SMS sent to a specific service number;
- Third Party Call, to set up and control calls.

In the "CallBySMS" service scenario different policies can be applied. Examples for different operations are:

StartSMSNotification:

- Time Policy: is the request time allowed to start the application? E.g. the notifications can be started from 8 am to 4 pm.
- Frequency Policy: how many times the operations can be invoked in a certain time frame, e.g. the start of notification can be handled only twice a day;
- Address Policy: are the parameter values allowed (e.g. is the service number associated to the application a valid number, the SmsActivationNumber should be a largeAccount number)?

NotifySmsReceptionRequest/Response:

- On this operation, the Time Policy, related to the time range in which the notification is sent to the Application, and the Frequency Policy are obvious policies to be applied on the base of the subscription between the Application and the Parlay X WS (i.e. Network Operator).

- The policy (of the category Address) on SenderAddress can be used to hide the SenderAddress (if the hiding of Address was established). This means that the SenderAddress has to be mapped with an opaque-id before being notified to a 3rd party application.
- The Session Policies that can be applied in relation to the *Correlator* parameter are Validity Time (i.e. evaluate the validity time within the scope of a Start/Stop Notification), and the so-called NumberOfOperationForSession: control on the number of notification in the session, defined in the subscription data.

MakeACall:

- Frequency Policy: the operation is violating the maximum frequency request? (e.g. the max frequency allowed for "A" to invoke MakeACall is 10 times in a month).
- Address Policy: are the operation addresses involved allowed?,Is the operator involved in the operation a valid one?
- User Policy: is it violating any privacy constraints on users (e.g. can "A" call "B")?

7.1 Description of the approach of automatic code generation for enforcement of policies

A reference model for the specification of policies addresses various aspects of controlling the usage of service capabilities. There is a difference between the authentication/authorisation policies and the usage policies. The former define the rules for identifying and granting the access to a calling application and are not considered here. Instead, the usage policies define the conditions and constraints on the invocation of services, on the data exchanged, on the frequency and timing of the access, etc.

These rules may be defined for all the applications that are subscribed for the controlled resource, for a particular application and even for a subset of the application functionality defined, e.g., by the usage session. The assignment of the policy sets to an application is performed at the subscription time, when the SLA between the application and the service provider is established.

The goal of the presented activity is the automated generation of the code for the enforcement of policies associated to the application by the SLA. Once generated, the policy handlers are deployed at the middleware level and control the access to the service capabilities through interception mechanism. More precisely, the control is performed as follows.

- During the authentication/authorisation phase, the application is identified and the corresponding identity values are extracted, including, e.g., application ID, subscription ID, and session IDs.
- The call is forwarded to the policy management gateway, where all the policies associated to the specified identity values are enforced.
- If the control of all the policies succeeds, the call is propagated to the target service, and the related policy control parameters are updated.
- If the control of one of the policies fails, the access to the resource is denied, and optionally a corresponding fault message is returned to the application.

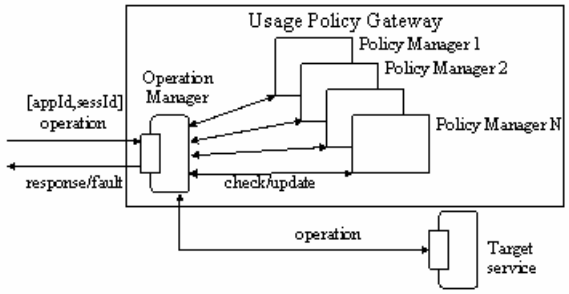

Figure 10: Policy Management Architecture

The approach of automatic policy enforcement code generation relies on the following architecture, implemented as a set of BPEL processes (Figure 10). The architecture consists of two layers. In the first layer, the *OperationManager* process (OM) receives an operation call and sends a "check operation" message to all the *PolicyManager* processes (PM) associated to the application. These processes represent the second layer

and are used to store, update and check policy measurement parameters associated to an application, a session, or a user. The PM processes evaluate the policy condition against the policy parameters or message parameters, and provide a response to the OM process. In case of policy violation, the fault message is sent to the application. Otherwise, the PM processes are called to update corresponding measurements (e.g., increase the counter of operations per day), and the call is propagated.

The generation of the BPEL processes implementing this architecture is performed as follows. Depending on subscribed services/operations, a set of OM process definitions are created. The set of generated PM processes depends on the set of application policies and their type. In particular, a separate PM process definition is created for the group of application-related policies (e.g., frequency, time, addressing); for the policies related to a particular session type; for the user-related policies. See the annexes for a more detailed description of the implementation and for the BPEL code examples.

7.2 Description of the approach for policy negotiation through constraints solving

A constraint-based approach for specifying and enforcing policies is proposed. The model combines basic features of process calculi [16] and concurrent constraint programming (*cc programming)* [17]. Specifically, a process calculus that extends cc programming by adding synchronous communication and by providing a treatment of local names à la pi-calculus rather than as variables with existential quantification is defined. The considered constraints are *soft*, i.e. they extend classical constraints to allow the representation of preference levels. In the resulting framework, policies on the capability of a web service are expressed as constraints and the parties wishing to define a policy are modelled as communicating processes of the model. A single process P= tell c.Q can place a constraint `c` corresponding to a certain policy and then evolve to process `Q`. Alternatively, two processes $P = \overline{p}\langle \widetilde{x} \rangle.P'$ and $Q = p\langle \widetilde{y} \rangle.Q'$ that are running in parallel (P|Q) can synchronise with each other on the port *p* by performing the output action $\overline{p}\langle \widetilde{x} \rangle$ and the input action $p\langle \widetilde{y} \rangle$, respectively, where $\widetilde{x}$ and $\widetilde{y}$ stand for sequences of names. Such a synchronisation creates a constraint induced by the identification of the communicated parameters $\widetilde{x}$ and $\widetilde{y}$, if the store of constraints obtained by adding such a new constraint is *consistent*, otherwise the system has to wait that a process removes some constraint (action `retract c`). The restriction operation (x) is used to define a local name x, thus allowing for local stores of constraints. Synchronising processes may have the effect of combining their respective local stores. This constraint-based model is applied in [1] for specifying and enforcing the following policies on time and frequency of the CallBySMS service.

- **Time Policy.** Suppose the Third Party Application (TPA) requires that the initial time i and the final time f of a service must be respectively in the time intervals [6am,…,9am] and [4pm,…,6pm] and the cost k must be less than max_cost. This policy is represented by the constraint c1 = (6am ≤ i ≤ 9am) x (4pm ≤ f ≤ 6pm) x (k ≤ max_cost). The Telco WS (TWS) guarantees that the initial time is in [8am,…,10am] and the final time is in [5pm,…7pm] and the cost of the service is k= cost(f-i), where the function 'cost' specifies that the cost of the service is 10 per hour for the first 5 hours, while it is 5 per hour for the remaining time. The constraint corresponding to this policy is c_2 = (8am ≤ i ≤ 10am) x (55pm ≤ f ≤ 7pm) x (k= cost(f-i)) .
- **Frequency Policy**. TWS offers a frequency *fr* that depends on the time interval *ti* according to a function freq(ti) which holds 6 if ti ≤ 5, while it holds 3 otherwise. This policy is specified by the constraint c_3 = (fr=freq(ti)). TPA does not pose any policy on the frequency.

TPA and TWS are defined as processes running in parallel. In order to check whether the time policy is enforced, one needs to equip the current model with a notion of time. Thus, a process $Clock_T$ is introduced that simulates the actual time by increasing a time variable *t*. The formal specification is as follows.

$$TPA(p,q,t) = (i)(f)(k)(fr)(s)\,tell\ c_1.(\overline{p}\langle i, f, k\rangle.\overline{s}\langle\ \rangle \mid \overline{q}\langle f - i, fr\rangle.s\langle\ \rangle.\ PE(i, f, t))$$

$$PE(i, f, t) = check\ (i \le t \le f).\ start$$

$$TWS(p,q) = (i)(f)(k)(t)(fr)\,(tell\ c_2.p\langle i, f, k\rangle \mid tell\ c_3.q\langle t, fr\rangle)$$

$$Clock_T(t) = retract\ (t = T).\ tell\ \ (t = T + 1).Clock_{T+1}(t)$$

$$System = (p)(q)(t)\,(TPA(p,q,t) \mid TWS(p,q) \mid Clock_0(t) \mid tell\,(t = 0))$$

Intuitively, TPA and TWS place their own constraints and then TPA tries to synchronise with TWS on port *p* and on port *q*. The synchronisations succeed because the resulting combinations of constraints c_1, c_2 and c_3 are consistent. This amounts to saying that TPA and TWS have concluded a SLA contract on time and frequency. As a consequence, TPA evolves to the process PE, which is in charge of executing an instance of the service and enforcing the policies. In particular, PE first verifies that the actual time *t* is in the admitted range (action check $i \leq t \leq f$) and starts the execution (action *start*).

The detailed description is provided in the annexes, presenting the description of the CallbySMS use case extended with negotiation aspects, a paper accepted at TASE'07.

8 Transactions for handling exceptions in service composition

To discuss the issues related to transactionality, a scenario is considered that consists of a content provider that makes it possible to buy the rights to broadcast certain events, e.g. concerts and parliament meetings, and a media provider that makes it possible to broadcast audio streams on the Internet and to a set of registered portable phone numbers. Having a combined "book and broadcast event" service is extremely useful, if not necessary, for an event broker, hereafter called User, whose intent is organising the audio broadcast of events.

Thus, the scenario involves four interacting partners: User, Book&Broadcast (B&B), MyEvent and MyBroadcast services. The User interacts exclusively with the B&B, which is responsible for the negotiation with the pre-existing MyEvent and MyBroadcast services, so all partners agree on their mutual needs and constraints. In particular, the case study describes the behaviour exposed by B&B that allows the user to book the rights to an event, as well as the required bandwidth to broadcast it. In this setting, the User must be able to submit a request for an event rights and for a broadcasting option, change it, accept or reject the offer provided, or stop the procedure by sending a cancellation message.

The B&B service model is supposed to provide this functionality interacting with the MyEvent and MyBroadcast components. Each of these two components must provide the possibility to accept a request (for an event and for a broadcasting bandwidth respectively), answering positively or not, and making it also possible to confirm or disconfirm the operation. While this can be hard-coded in a complex piece of (BPEL) code that handles explicitly all cases, a much simpler and more neat way to express this is to exploit a transactional specification, i.e. to say that the two component services export a 'booking' method, and that such a method is transactional, i.e. it supports primitives to commit, to roll-back, to compensate or to handle exceptions. Business processes usually require long-running transactions for which compensations mechanisms replace traditional locking, check-pointing and rolling-back. In fact, in many applications, the initial situation cannot just be re-established with a roll-back, instead some counteraction must be activated. The simplest example is the late cancellation of a reservation, for which some fees have to be paid.

A relevant aspect of this scenario regarding transactionality is that the establishment of a consistent situation where the whole process of the B&B sale is completed or rolled back cannot be associated to the behaviour of a single service, but requires operating on the various partners in a sound and coordinated way. In particular, while a variety of situations may take place, only two cases are admitted: either the sale is completed, in which case the User has bought the rights to the event and the broadcasting bandwidth, and the MyEvent and MyBroadcast services have successfully completed their transactional activities, or the sale is refused, in which case only situations which are consistent under a global perspective are admitted, i.e. rule out cases where some partner has nevertheless committed a portion of the transaction.

8.1 Description of the approach (BPEL + SAGAs)

The approach to handle transactionality in the composition of services starts from a situation where the components comes with a notion of transactional contexts, which define the transactional aspect of the functionalities they provide. In the simpler scenario, both the functionalities, 'book an event' and 'book a bandwidth', must be perceived as transactional, i.e. as logically atomic, and therefore need to be defined within two respective transactionality contexts *Event_booking* and *Bandwidth_booking*. Notice that in more complex scenarios, a single service may feature more than one transactional context, and it may provide a variety of transactional functionalities. To link this notion to the BPEL web service language, the transactional contexts are represented by means of BPEL scopes that embed question/answer protocols. Figure 11 represents the

MyEvent transactional service: different BPEL scopes are used to define the transactional activities (left) and their rollback (right).

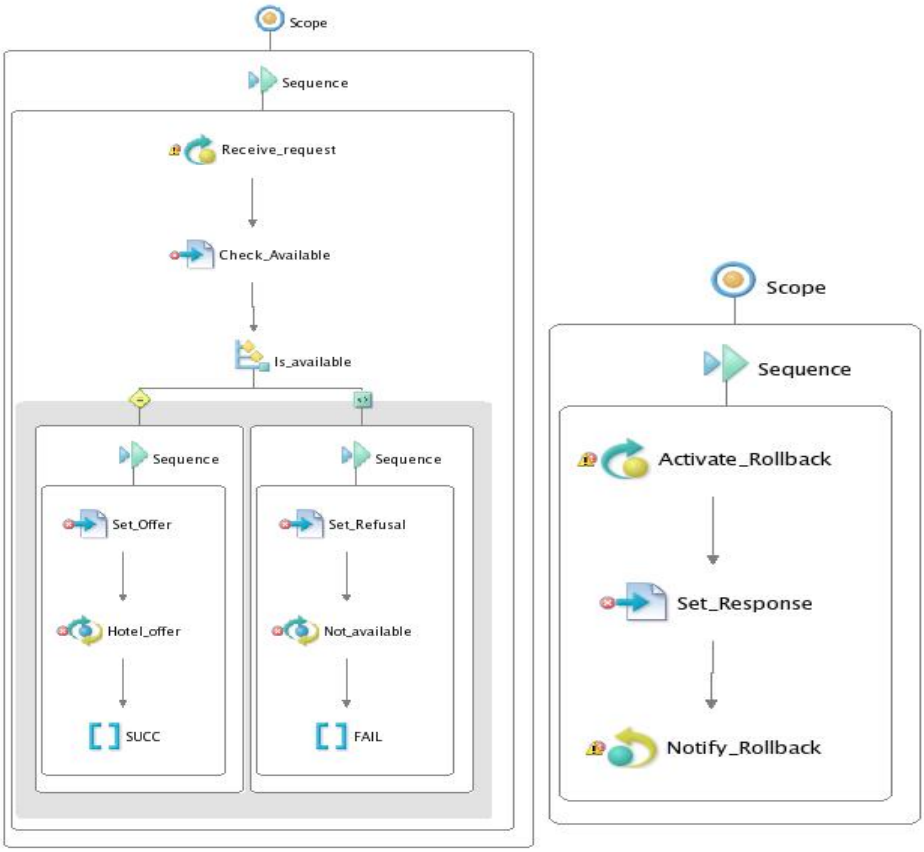

Figure 11: BPEL Process

Given this, the way transactionality must be handled in the composite service is specified by a transactional aspect of requirements, referring to the composition of transactional contexts. For this purpose, only existing powerful formal languages are used with a clear semantics, so to be able to automatically convert transactionality requirements into composition constraints. In particular, the SAGAS language [1] is considered, developed at the University of Pisa with the aim to clearly state transactionality requirements of composed processes. SAGAS is a structured language for compensable orchestration, originally defined as a process algebra, but also available as an XML schema. A prototypical implementation is available that includes a GUI for the design of transactional business processes, an interactive simulation environment, and a distributed orchestration engine for execution. SAGAS features a compact set of constructs that include sequential and parallel composition, nesting of transactions, forward and backward recovery strategies. Various flavours of SAGAS have been formalised, which by semantic variants allow a more or less liberal handling of parallelism in the compensation of processes.

When applied to the B&B scenario, this amounts to specifying that, for instance, the B&B carries out both the two transactional activities “book an event” and “book the bandwidth for broadcasting it”, in some order., i.e. in SAGAS jargon, that Event_booking * Bandwidth_booking.

The SAGAS specification, together with the semantics of the chosen transaction model, is what allows for the automated extraction of the Coordinator entity that links together the various partners; in the considered scenario, for instance, this means that upon receiving a refusal from the User, the B&B will contact the Coordinator with a ‘request for cancellation’ that will trigger, in turn, the cancellation for either the MyEvent and/or the MyBroadcast (depending on which of them has been made active already).

Transactionality is only one of several possible aspects involved in a composition of web services. For instance, the dependency between data values distributed amongst different components is a different aspect, and the mutual consistency of the logical choices put in place by different services is another one. Each of these aspects, and possibly more of them, need to be taken into account into the composition, either by properly stating them as an aspect of the composition requirements, or by suitably programming them directly into the

composed service. The aforementioned approach envisages the description of composition requirements decomposed into different, independently specified aspects, each one referring to a specific feature that must be presented by the composed orchestrator.

Note that the model of transaction is defined by an independent entity, which defines the primitives available to handle a transaction (e.g. commit, cancel, roll-back) and their semantics. This definition is essential both to define the behaviour of transactional components, and to enforce transactionality in the composition. Several standard models of transaction are available, e.g. Business Transaction Protocol (BTP), WS-Atomic Transaction, WS-CAF; by not committing to any of them, the aforementioned approach is decoupled from a specific notion of transaction and can suitably be integrated in different settings. The intent is to develop a suitable LTL logic and verification tools for SAGAS that could serve to establish a weaker notion of conformance between SAGAS and its BPEL realisation (e.g. w.r.t. a given scenario or a logical specification).

9 References

[1] C. Moiso, L. Sarchi, E. Morello, P. De Lutiis, M. Bonifati - Identity Federation for services in convergent networks. In Proc. of ICIN'06, 2006, 109-114.

[2] http://www.projectliberty.org/index.php/liberty/specifications__1

[3] OASIS Security Services (SAML) TC - Security Assertion Markup Language (SAML) - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[4] SOAP 1.2: http://www.w3.org/TR/soap12-part0/

[5] 3GPP - Open Service Access (OSA) – Parlay X Web Services (Release 6), TS 29.199 v6.x.y

[6] L. Ferro, E. Morello, E. Thuegaz - Multimedia conference specification - TILab Document Cod. Doc. DPC 2006.01824

[7] M.H. ter Beek, S. Gnesi, F. Mazzanti, C. Moiso - Formal Modelling and Verification of an Asynchronous Extension of SOAP. In Proc. of ECOWS'06, IEEE Computer Society, 2006, 287-296.

[8] Sensoria IST Project – Deliverable 5.1a - Process calculi and coordination languages with costs, priority and probability.

[9] C. Moiso - Interaction patterns between Telco composed services and component services. TILab Document Cod. Doc. DPC 2006.00858.

[10] M.H. ter Beek, C. Moiso, M. Petrocchi - Towards Security Analyses of an Identity Federation Protocol for Web Services in Convergent Networks. In Proc. of AICT'07, IEEE Computer Society, 2007.

[11] M.H. ter Beek, C. Moiso, M. Petrocchi – Modelling and Analysing an Identity Federation Protocol: Federated Network Providers Scenario. In Proc. of YR-SOC'07, 2007, 94-100.

[12] F. Martinelli - Analysis of security protocols as open systems. TCS 290, 1 (2003), 1057-1106.

[13] C. Moiso - Service Oriented Architecture for Telco Service Layer presented at Bridging Global Computing with Grid (BIGG), Nov. 29th, 2006 , TILab Document DPP 2006.01974.

[14] J.L. Fiadeiro, A. Lopes, L. Bocchi - The SENSORIA Reference Modelling Language: Primitives for Service Description and Composition. SENSORIA Research Report, 2006. (Short version: A Formal Approach to Service Component Architecture. In Proc. of WS-FM'06, LNCS 4184, 193-213.)

[15] L. Bocchi, J.L. Fiadeiro, A. Lopes - The SENSORIA Reference Modelling Language: Primitives for Configuration Management. SENSORIA Research Report, 2007.

[16] M.G. Buscemi, U. Montanari - Cc-pi: A constraint-based language for specifying service level agreements. In Proc. ESOP'07, LNCS 4421, 2007.

[17] V. Saraswat, M. Rinard - Concurrent constraint programming. In Proc. of POPL'90, ACM Press, 1990, 232-245.

[18] M.G. Buscemi, L. Ferrari, C. Moiso, U. Montanari - Constraint-Based Policy Negotiation and Enforcement for Telco Services. To appear in Proc. of TASE'07.

[19] R. Bruni, H. Melgratti, U. Montanari - Theoretical foundation for compensation in flow composition language. In Proc. of POPL'05, ACM Press, 2005, 209-220.

10 Annexes

This is the list of annexes to this deliverable:

A high-level description of the Sensoria Telecommunication Case Study is part of a presentation of Telecom Italia at BIGG 2006 Workshop [Del81a_ch2_BIGG - Moiso v1 0.pdf].

In relation to the chapter about Service Modelling, two documents are provided: the "Call and Pay Taxi" Service Scenario [Del81a_ch3_Call&PayTaxiSensoria.pdf] explaining the service in all the details, and the "Call and Pay Taxi" Service Scenario modelled using SRML [Del81a_ch3_SRLMcallpay(v1 0).pdf].

Regarding the analysis of a protocol for identity federation are provided: one paper presented at AICT'07 [Del81a_ch4_full-idp.pdf], one presented at YR-SOC'07 [Del81a_ch4_final-fnp.pdf] and a Telecom Italia paper presented at ICIN'06 [Del81a_ch4_Token Injector ICIN 2006 v1.0.pdf]

Related to the formal modelling and verification of an asynchronous extension of SOAP one paper, presented at ECOWS'06, is provided [Del81a_ch5_ecows06.pdf]

On the synchronous and asynchronous interaction, three documents are provided: a Telecom Italia analysis [Del81a_ch6_asincronicitv1.pdf], a multi media conference service specified by Telecom Italia [Del81a_ch6_MMConferenceSpecification.pdf] and a BPEL approach on the theme with an article submitted to ICWS'07[Del81a_ch6_ICWS07paper.pdf]

In relation to Policies for exposing Web Services are provided: a detailed analysis of policies [Del81a_ch7_Wp8sensoriapolicycasestudyv1.pdf] and [Del81a_ch7_PolicyforD81a-v1.pdf], an article accepted at TASE'07 [Del81a_ch7_TelcoCaseTASE2007.pdf], and a policy enforcement approach [Del81a_ch7_PolicyEnforcv1.pdf].