Designing and Deploying Service-Centric Systems: The SeCSE Way

The SeCSE Team
c/o Engineering Ingegneria Informatica
Via S. Martino della Battaglia, Roma, Italy

ABSTRACT

In spite of recent research, designing and deploying large-scale service-centric systems remains a challenge. The SeCSE project tries to tackle this challenge by creating new methods, tools and techniques for requirements analysts, system integrators and service providers that support the cost-effective development and use of dependable services and service-centric applications. The results achieved in the project are being experimented in the European automotive and telecommunication sectors. In this paper we presents the demo that we have been developing for the last two project years. The goal is to show the tools available to a service integrator that creates a new service relying on others that are already existing and are offered either by the project partners or by other players in the Internet. The paper also describes how this resulting service behaves at runtime.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Domain Specific architectures, Data mapping, Distributed objects

1. INTRODUCTION

In spite of recent research, designing and deploying large-scale service-centric systems remains a challenge for system designers and integrators, a challenge that the EU-funded integrated project called SeCSE (Service-Centric Systems Engineering) is tackling [1].

SeCSE's mission statement is to create new methods, tools and techniques for requirements analysts, system integrators and service providers that support the cost-effective development and use of dependable services and service-centric applications in the European automotive and telecommunication sectors. The four-year research project covers four main activity areas:

Service engineering: how to model, specify and publish services in service registries to be discovered, composed, invoked and monitored in service-centric systems;

Service discovery: how to find and retrieve services from service registries, at design time, so that these services are compliant with requirement and architecture models, and at run-time, so that it is possible to find alternative services to invoke in a service-centric application;

Service-centric systems engineering: how to create service compositions able to reconfigure themselves at runtime based on their execution context and on the status of services they are using;

Service delivery: how to expose services in a distributed environment, monitor whether services comply with requirements, and therefore support system reconfguration in response to service failure or divergence from these requirements.

SeCSE methods, techniques, and software tools are experimented in the automotive and telecommunication sectors through involvement of some industrial partners.

2. THE SECSE METHODOLOGY

Supported by a reference scenario, SeCSE is presented in this paper as an integrated working environment allowing the users to develop service-centric systems.

The steps that so called *Service Integrators* [6] should follow encompass several activities, ranging from the definition of its business requirements to the publication of these in a federation of heterogeneous service registry:

1. *Business requirements definition.* This is an iterative phase allowing the stakeholders to provide in some form the business requirements (functional and non functional) expected for a service-based system. A set of services somehow fitting with the provided requirements are retrieved. The description supporting these services will be used by the users to refine their requirements and enable more accurate discovery. This set of services will be the input for phase 2.
2. *Composition.* The services obtained by the previous

phase must be now composed by means of a BPEL-like ([3]) workflow. These services will be described as abstract specifying only their functional behaviour. A discovery tool named Architectural Service Discovery will find those concrete services that, matching the specified functional behaviour, can replace the abstract services. The Service Integrator can select one or more concrete services for each related abstract one to be then dynamically bound at run-time. Moreover, he/she can define the rules and the constraints that will determine at runtime the binding to specific services. This binding can depend on various aspects including the context in which the composition final user operates, the possible faults caused by the used services, the publication of new, appealing services.

3. *Instrumentation - Monitoring rule definition.* Once the service centric system has been designed (phase 2), a policy will be set to monitor that the execution works as expected. If an undesired situation is detected, the rules defined in phase 2 will be fired.

4. *Service regression testing.* Anytime a service integrator will acquire or use a service necessary for its composition (see phase 2), he/she will have the opportunity to test the corresponding functionality by means of regression testing tools to check that what the service has declared in its interface, and, possibly, agreed with the service-centric system is actually respected.

5. *Deployment of the service composition.* The service centric system is deployed to the run-time engine to be executed.

6. *Service Centric system description.* Once deployed, the service centric system is described. This is done with a facets approach providing a mechanism to describe services (simple and composite) from different perspectives leaving to the user flexibility margins. A set of predefined facet is provided for general service description, functional description, the QoS properties, commercial aspects, non functional behaviour and the test cases.

7. *Service Centric system publication.* The latest steps concern the publication of this service description. It will be possible to publish this service in a SeCSE Registry taking part to a federation of heterogeneous registries linked by replication policies. The SeCSE registry is an XML registry able to host the service description based on facets.

When all of these steps are successfully terminated (of course the development process may not follow a waterfall model), the service-oriented system is ready for execution. At runtime the system will be able to evolve by changing the bindings to the external services on the basis of the rules that have been defined through its design and instrumentation. In the following of the paper we provide more details on the design time and runtime aspects by referring to the specific example that we propose to demonstrate.

3. DEMONSTRATION SCENARIO

The following demonstration scenario was provided by the pilot users of the SeCSE project, operating in the automotive and telecom domains.

Roberto, a system integrator at Company X, intends to implement a BusinessTrip (composed) service, to help users with planning a car trip according to their appointments. In particular, the service should be accessible from the car, so that appointments in agenda can be confirmed or changed, based on the current geographical position of the car. In case the next scheduled appointment is not feasable, the service should automatically open a telephone call with a secretary of the business man to rearrange the agenda. When the driver arrives at his destination, the service should suggest him the nearest parking lots that have free spaces.

In the following, we first show how the SeCSE tools are used by Roberto in the activities of the service development process, according to the methodology described in Section 2, and then illustrate some usage scenarios by different users to highlight the self-(re)configuration capabilities of the composition, enabled by the run-time tools.

3.1 Design Time

3.1.1 Requirements analysis

At this stage, Roberto can form queries from his requirements specification to discover services that are related to the requirements in some form. Descriptions of these discovered services are retrieved and explained to him, then used to revise and refine his requirements specification to enable more accurate service discovery. But let's see how this works. SeCSE's iterative and incremental requirements process is supported with a suite of software components. The UCaRE component provides Roberto with two main capabilities. Firstly, it enables Roberto to specify UML use case specifications and VOLERE requirements [9] on systems that might or might not be implemented using services. It extends UML use case specifications with requirements expressed using the VOLERE shell. Both use cases and requirements are specified in structured natural language, consistent with how Roberto specifies requirements in the Rational Unified Process. Secondly, it enables Roberto to generate service queries directly from use case and requirements specifications using the simple tick boxes demonstrated in Figure 1. Once Roberto created a service request, UCaRE passes it to EDDiE, SeCSE's service discovery engine. After some manipulation on the natural language (see the paper by Zachos *et al.* [11] for more details), query matching is in two steps: (i) XQuery text-searching functions to discover an initial set of services descriptions that satisfy global search constraints; (ii) traditional vector-space model information retrieval, enhanced with WordNet, to further refine and assess the quality of the candidate service set. This two-step approach overcomes XQuerys limited text-based search capabilities. Services retrieved from the registry are reported to Roberto through the Service Browser component shown in Figure 2.

3.1.2 Design

Based on the results from the requirements analysis phase, Roberto identifies the activities for the Business Trip and

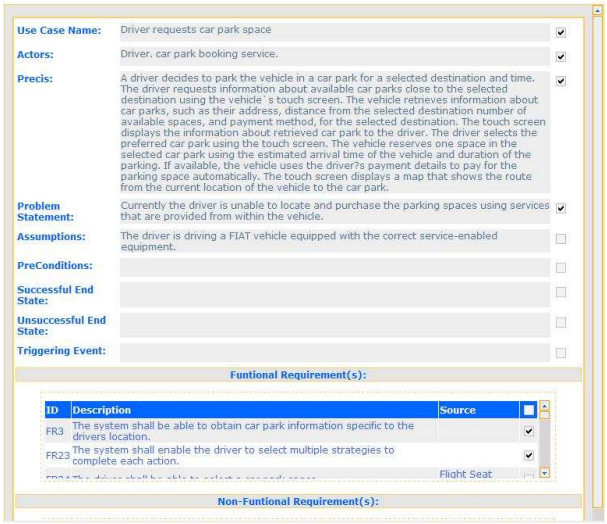


Figure 1: UCARE GUI

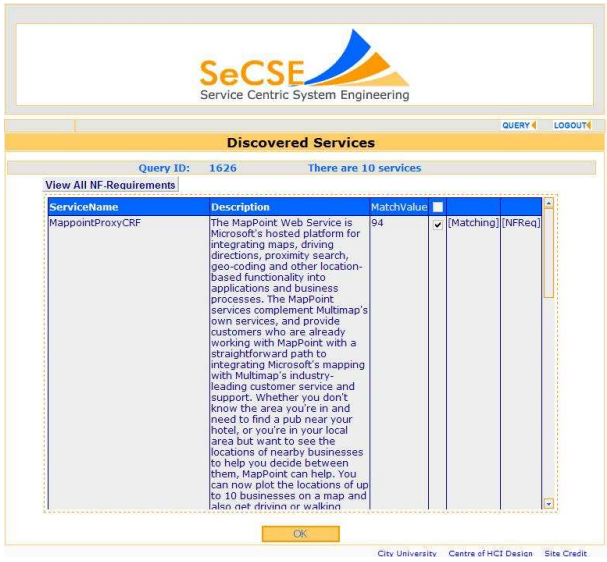


Figure 2: Service browser

draws a workflow model as in Figure 3. Indeed, he realizes that he needs the following service operations:

- *findDuration* to compute the distance of two geographical points;
- *calculateTime* to compute the time to cover that distance by car;
- *checkSchedule* to get the next appointment in agenda;
- *makeCall* and *getCallInformation* to automatically open a telephone connection between the user and his/her secretary, and to check when the call ends;
- *sendSms* and *getSmsStatus* to send a notification to the user of appointment confirmation, and to check that the sms is actually sent;
- *PointInPoly* to know the current position of the car;
- *findParking* to find the list of the nearest parkings.

Then Roberto starts using the Composition Designer tool (CD) to focus on the low level design of the system. In particular, he finalizes the interfaces of these services in WSDL documents and sketches the interaction with them in terms of BPEL fragments. From these data the CD generates a UML model that is recognized by the SeCSE Architecture-time Service Discovery (ASD) tool (described in the paper by Zisman and Spanoudakis [12]) as a query to search for services in the registry that match. Roberto could iteratively refine the search to get results with higher precision. In this case, Roberto prefers to have a wider, even if less precise, view of the services available so that he can eventually modify/adapt the model at functional or just syntactic level, according to what he is getting. Figure 4 shows the list of services returned by the ASD tool for the service operation *checkSchedule*.

In order to both make the Business Trip service adaptable to different user preferences, and increase the overall quality, e.g., availability and reliability, Roberto finds quite convenient to enrich the description of his service with binding and monitoring rules, and constraints and preferences, provided by the SeCSE languages.

Dependency constraints specification. Roberto indicates that the operations *makeCall* and *getCallInformation* should be provided by the same service as they are state-dependent. This also applies to the operations *sendSms* and *getSmsDeliveryStatus*.

Service list preferences definition. For the demostration's purpose, the SeCSE registry has been populated with a number of services. In particular, those used in the proposed scenario are the following (a service is real wherever its provider is explicity said):

trip duration: XnavigationCEFRIEL, XnavigationULISSE, and TimeServiceCEFRIEL;

agenda management: CalendarServiceEMIC (offered by Microsoft), and CalendarServiceCEFRIEL (built with API from Google);

phone call: ThirdPartyCallTLAB (operated by Telecom Italia Lab), ThirdPartyCallCEFRIEL, ThirdPartyCallPHONE, and ThirdPartyCallIND;

sms: SendSmsTLAB (operated by Telecom Italia Lab), SendSmsCEFRIEL, and SendSmsPHONE;

car position: MapProxyCEFRIEL, and MapProxyCRF (operated by CRF);

car parking search: Microsoft Mappoint, and PointOfInterest (operated by CRF).

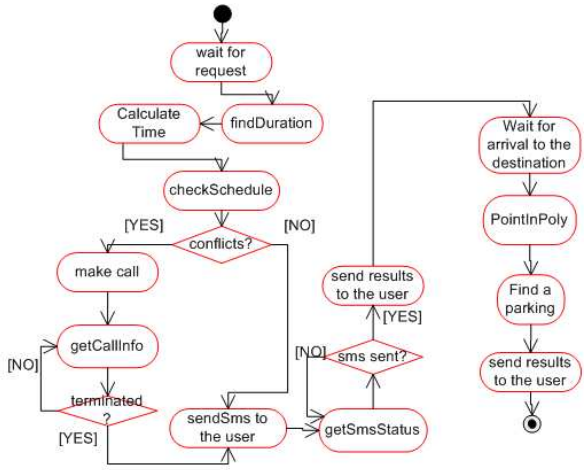


Figure 3: The Business Trip workflow

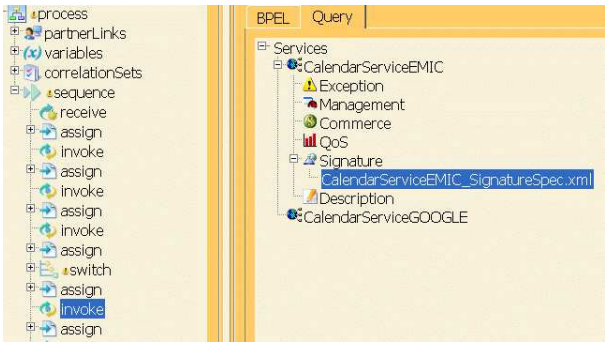


Figure 4: The ASD result for checkSchedule

From the results of the ASD search, Roberto indicates a subset of these services in a preference list, which will be used to select the bindings at run-time. The other services will be discovered through the RSD tool as explained later.

Binding rules definition. These are Event-Condition-Action (ECA) expressions that regulate the bindings selection at run-time. The event part may be, for example, of type *ActivityBinding*, which means that the rule is triggered when the activity of the process, specified in the rule, is not bound. In this case, the action part contains the binding to be applied if the condition of the rule is evaluated to true. Another type of rule is related to a property violation event, to specify what to do in that case, for example start some recovery action. More details on the types of rules and the language to express them are in the paper by Colombo *et al.* [7]. Roberto defines the following rules of *ActivityBinding* type:

checkSchedule: (E: ActivityBindingEv; C: true; A: Bind to the agenda in the user info), i.e., the binding should be with the agenda service indicated by the user;

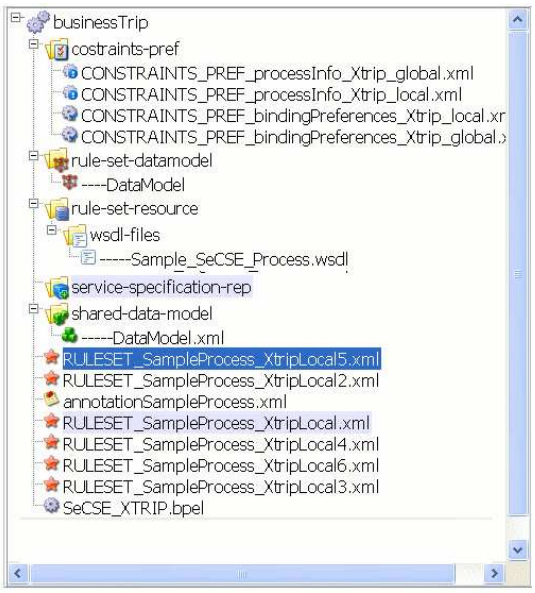


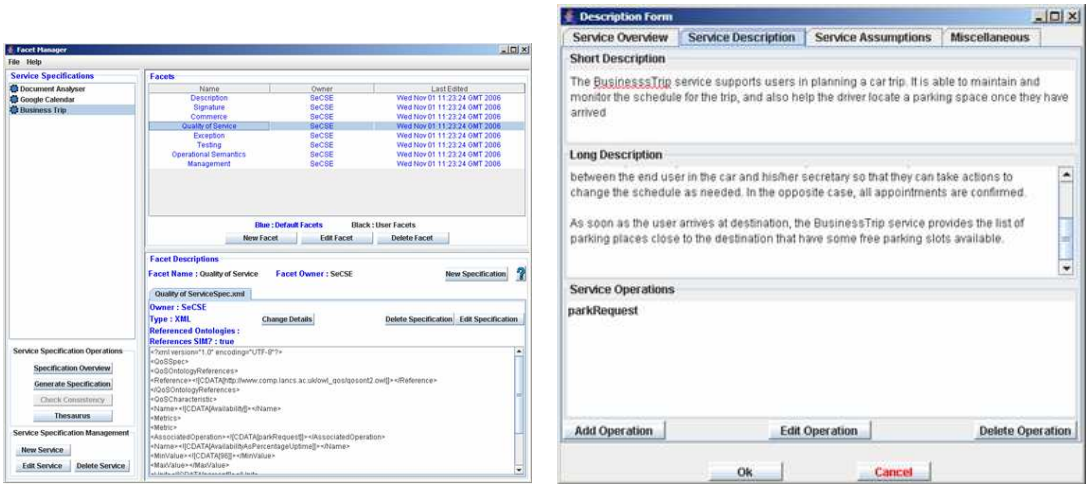
Figure 5: The composition design artifacts

makeCall and *sendSms*: (E: ActivityBindingEv; C: provider = bestPrice; A: select the best price provider from the list of variants), i.e., the best price telecom operator (among those already available to the system) should be selected for phone services; and (E: ActivityBindingEv; C: provider = bestPrice; A: select the best price provider using the RSD), i.e., the best price telecom operator should be searched through the SeCSE Run-time Service Discovery (RSD) tool and bound to the related activities.

findParking: (E: ActivityBindingEv; C: destination = Torino; A: Bind to PointOfInterest), and (E: ActivityBindingEv; C: destination != Torino A: Bind to Mappoint), i.e., the service PointOfInterest has to be used to look for parkings in Torino, and the service Mappoint in other locations.

Monitoring rules definition. Roberto realizes of a need for a precondition on the *makeCall* operation, related to the input phone number to be valid, and for a postcondition on the *findDuration* operation concerning the output value to be greater than 0. These pre/post conditions, expressed in SECMOL, the SeCSE monitoring language [2], will be verified at run-time by the SeCSE monitoring sub-system (described in the paper by Baresi and Guinea [4]) and events on their violation handled as specified in the binding rules discussed above.

Global QoS constraints definition. In order to make the Business Trip service more appealing to users, Roberto defines a global constraint to guarantee that the price will not exceed some value, e.g., 1.4 euros and that bindings will be selected so that, at the end, the response time will be the least possible. For this, Roberto needs to indicate the probability of execution for each branch of the switch construct of the workflow, and estimations for the number of iterations of each loop. So, he decides that most probably (60%) there won't be conflicts in agenda, and that at most



(a) Faceted Specification Tool

(b) Description Facet Form

Figure 6: Business Trip Specification

3 checks will be made for end of call, and 2 for sms sent. This data is to allow the system to make estimations of the QoS of possible global bindings of the workflow just before execution. The estimation of the one selected will be refined during execution, according to the monitored QoS of the executed services and to the actual process data.

This configuration of the system will be alternative to that described by the (local) binding rules specified earlier, as the run-time selection of services will be based on QoS aspects only, and without considering context information. Details on the global binding mechanism are in the paper by Di Penta *et al.* [8]. The global QoS constraints automatically define monitoring rules (expressed as pre-conditions for all the invoke activities of the process) to check their respectance during execution. Also, they define a (global) re-binding rule, where the trigger event is the precondition violation by the monitoring system, and the recovery action is re-binding of the process part still to be executed.

System deploy. All the artifacts produced by Roberto are shown in the project view of the CD, as in Figure 5. After completion, Roberto may run the automatic deploy function of the CD. As an effect: proxy services (to enable dynamic binding) are generated from each WSDL interface he had previously defined (and used as input to the ASD tool); the proxy services are statically bound to the BPEL process; and the composed service is automatically deployed in the SeCSE run-time environment. All of these operations happen behind the scenes, so Roberto only sees that the Business Trip service is now in place to be published and executed.

3.1.3 Specification

Roberto proceeds to create a corresponding specification for the Business Trip service, using the SeCSE Faceted Service Specification approach [10]. This is a mechanism that supports the bringing together and ordering of specifications (expressed in different schemes or languages) that address a range of service properties.

Each facet focuses on one or more service properties (e.g. general description, binding, etc). For example, an Operational Semantics facet may embed OCL or BPEL based specifications, or both if desired, that describe service behaviour. By also supporting the use of third party specification mechanisms, the Faceted Service Specification approach can maintain compatibility with other approaches, and both current and future developments. Currently in SeCSE we support the following facet types: Signature, Description, Operational Semantics, Exception, QoS, Commerce, Testing and Management; each which can be populated with corresponding specifications.

In order to create the faceted specification, Roberto makes use of the SeCSE specification tool which allows to him to create, specify and manage the facets within his specification. Figure 6 (a) shows the specification tool being used to create the Business Trip service specification. In the top right a table displays the facets that exist within the specification, below this is a preview pane that can be used to view the specifications within a selected facet. The tool provides a set of forms that guide Roberto through the specification process. Figure 6 (b) shows the form that Roberto used for the Description facet specification - this form has been specifically designed to support the process of Requirements Based Service Discovery as described in sub-section 3.1.1. Roberto used a similar approach for specifying the other facets and then proceeds to publish his completed Business Trip service specification.

3.1.4 Publication

Roberto realizes that for the Business Trip users working in Spain the automatic selection of telephone service by some

(a) The Business Trip result

P.	Composition	Activity name	Concrete service endpoint
1	BusinessTrip_local	findDuration	http://secse.cefriel.it:8084/axis/services/xnavigationPort
1	BusinessTrip_local	PointInPoly	http://server.democefriel.it/mapproxy/MappointProxy.asmx
1	BusinessTrip_local	calculateTime	http://secse.cefriel.it:8084/axis/services/timeservicePort
1	BusinessTrip_local	FindNearby	http://findv3.staging.mappoint.net/Find-30/FindService.asmx
1	BusinessTrip_local	checkSchedule	http://server.democefriel.it/CalendarService/CalendarService.asmx
1	BusinessTrip_local	requestMeeting	http://server.democefriel.it/CalendarService/CalendarService.asmx
1	BusinessTrip_local	sendSms	http://secse.cefriel.it:8084/axis/services/SendSms2
1	BusinessTrip_local	getSmsDeliveryStatus	http://secse.cefriel.it:8084/axis/services/SendSms2

(b) The binding list for first scenario

Figure 7: The execution GUIs

local provider would be the right choice. This means that the service composition will also need to adapt to this new context. Unfortunately, the SeCSE discovery tools described earlier do not return any service for the *sendSms* operation from spanish telecom providers.

Before thinking of any recovery action, to modify the process in such a way it can be executed in the new context, the SeCSE framework offers another option to Roberto. Instead of considering only the *local* registry to search for the new service, he can consider the whole **publication infrastructure**, that is, the set of the 4 SeCSE registries that publish information about services. The publication infrastructure links the different registries into a single *abstract* SeCSE registry, where different (physical) registries can *subscribe* to the classes of services they are interested in and as soon as such information becomes available in the system, it is sent to the requesting registry. This means that each single registry does not contain the descriptions about all the services available in the network. This framework is described in the paper by Baresi and Miraz [5]. The local registry used by Roberto can issue a *subscription* to service *sendSms*, or to the class this service belongs to. The effect is that as soon as one of the registries publish a service description that fits Roberto's needs, this information is immediately dispatched to the *local* registry. Given the publish and subscribe policy adopted by the **publication infrastructure**, all the information about available services are re-transmitted after a user-defined lease time. This way, we can be sure that, at most after the lease time, any possible service description is propagated towards the local registry.

In the proposed scenario, Roberto's subscription is fruitful and, after the lease time, a new service that fits the query for the *sendSms* operation is found from the local registries. Thus: either Roberto adds a dedicated binding rule to the Business Trip description, or the spanish *sendSms* service will be automatically discovered (and bound) at run-time by the RSD tool, after the binding rule already defined for that operation fires.

3.2 Run Time

Other than the BPEL process execution, the SeCSE runtime environment provides the following functions: (i) **Dynamic selection** of the concrete services for the various activities of the service composition, according to the related user preferences and context information. Additionally, the RSD tool may be exploited to dynamically discover other services, if needed; (ii) **Monitoring** pre/post conditions surrounding some of the invoke activities of the process; (iii) **Recovery actions** activation in case of failure of some condition above.

In the sequel, we illustrate specific usage scenarios of the Business Trip, where these innovative funtions come into play.

3.2.1 Execution

We demonstrate the execution of the Business Trip by three actors, namely: John Smith, Roberto Tola and Mario Rossi, who use the service with different preferences and in different moments in time. Two graphical user interfaces are provided: one is for the Business Trip end-user to insert the input data and visualize the results on his on-board device; the other is for administration purposes, to show the bindings selected and used by the system in the various executions.

First Scenario. John Smith's preferences are as follows: MS Exchange is hosting his agenda; the best price telecom operator (among those already available to the system) should be selected for phone calls; his secretary is Matteo. From his car, John invokes the service by specifying that he is going from Milano to Torino. The service behaves as follows: the system calculates the duration of the trip; it dynamically binds to John's agenda; there are no conflicting appointments, so the trip is confirmed; at destination, the Torino parking service is exploited. Figure 7 (a) shows the result of the service execution as presented to John, i.e., the nearest parking lots in Torino, where he can find room for his car. Instead, the list of bindings selected and used by the system is shown in Figure 7 (b), labled with 1, and Composition type *BusinessTrip_local*, to mean that only local constraints and preferences are used. To be noted that the CalendarService hosted at Cefriel (which interfaces with MS Exchange agenda) is used, as well as the PointOfInterest service (MappointProxy in the uri), which is only able to look for parkings in Torino. Also, no phone call service is bound to the process as no appointment changes need to be made.

Second Scenario. Mario Rossi uses the same preferences as John Smith, but he invokes the Business Trip service by specifying that he is going from Milano to Orbassano (a small town close to Torino). Hence, the system calculates the duration of the trip and dynamically binds to Mario's agenda. There are no conflicting appointments, so the trip is confirmed. However, this time the sms service to be used is searched through interaction with the RSD tool. Also, at destination, the parking service provided by Mappoint is exploited, as this service also operates outside Torino. The new binding list for this execution is labled with 2 in Figure 8. To be noted that now the binding for the *sendSms* and *getSmsDeliveryStatus* activities has changed to the service provided by Telecom Italia operator.

Third Scenario. Roberto Tola's preferences are as follows: Google Calendar is hosting his agenda; Telecom Italia provider needs to be used for phone calls and sms; his secretary is Elisabetta. Roberto invokes the service while he is starting to drive from Milano to Torino. Thus, the system calculates the duration of the trip and it dynamically binds to Roberto's agenda. This time there are conflicting appointments, so the system establishes a phone communication between Roberto and his secretary. She moves some appointments, and an sms is sent to Roberto, notifying that his agenda has been updated. At destination, the Torino parking service is exploited. From Figure 8, list 3, one can see that now the activities *makeCall* and *getCallInformation* have been bound as they are actually required by the process, and the binding for *checkSchedule* has been changed to the service Google Calendar, as specified in the binding rule described in sub-section 3.1.2.

Fourth Scenario. At another time, Roberto Tola chooses a global preference setting where the total price of the Business Trip service execution is guaranteed to be within 1.40 euros and the response time the least possible. Also, now he is driving from Milano to Orbassano. So, the system first calculates the best configuration of services according to the global QoS constraints, then the process is actually started. The system calculates the duration of the trip, and it dynamically binds to Roberto's agenda. There are conflicting appointments, so a phone call needs to be made: at this point, the monitoring system detects that the price constraint could be violated as services considered with low probability at design-time are actually required (i.e., *makeCall* and *getCallInformation*). Therefore, rebinding is applied as a recovery action from the risk of not to be able to satisfy the global constraints. The system replaces some of the remaining services by choosing cheaper ones and relaxing the response time optimization constraint. At destination, the parking service provided by Mappoint is exploited. The new binding list is shown in Figure 9, where the replaced bindings are highlighted in dark grey. With the new bindings, the final price results of 1.39 euros, while the response time is slowed down of 60 seconds.

4. CONCLUSIONS

In this paper we have presented the results currently achieved by the SeCSE project through a specific example from the automotive domain. The example relies on existing services that are either offered by the project partners or by other players in the Internet.

1	BusinessTrip_local	PointInPoly	http://server.democefriel.it/mapproxy/MappointProxy.asmx
2	BusinessTrip_local	getSmsDeliveryStatus	http://163.162.173.70:8001/WsGateway/services/ppxSmsSend
2	BusinessTrip_local	findDuration	http://secse.cefriel.it:8084/axis/services/xnavigationPort
2	BusinessTrip_local	PointInPoly	http://server.democefriel.it/mapproxy/MappointProxy.asmx
2	BusinessTrip_local	calculateTime	http://secse.cefriel.it:8084/axis/services/timeservicePort
2	BusinessTrip_local	FindNearby	http://findv3.staging.mappoint.net/Find-30/FindService.asmx
2	BusinessTrip_local	checkSchedule	http://server.democefriel.it/CalendarService/CalendarService.asmx
2	BusinessTrip_local	requestMeeting	http://server.democefriel.it/CalendarService/CalendarService.asmx
2	BusinessTrip_local	sendSms	http://163.162.173.70:8001/WsGateway/services/ppxSmsSend
3	BusinessTrip_local	makeCall	http://163.162.173.70:8001/WsGateway/services/ppxSipTpc
3	BusinessTrip_local	FindNearby	http://findv3.staging.mappoint.net/Find-30/FindService.asmx
3	BusinessTrip_local	getCallInformation	http://163.162.173.70:8001/WsGateway/services/ppxSipTpc
3	BusinessTrip_local	requestMeeting	http://secse.cefriel.it:8084/axis/services/ServiceSoap
3	BusinessTrip_local	findDuration	http://secse.cefriel.it:8084/axis/services/xnavigationPort
3	BusinessTrip_local	sendSms	http://163.162.173.70:8001/WsGateway/services/ppxSmsSend
3	BusinessTrip_local	calculateTime	http://secse.cefriel.it:8084/axis/services/timeservicePort
3	BusinessTrip_local	getSmsDeliveryStatus	http://163.162.173.70:8001/WsGateway/services/ppxSmsSend
3	BusinessTrip_local	checkSchedule	http://secse.cefriel.it:8084/axis/services/ServiceSoap
3	BusinessTrip_local	PointInPoly	http://server.democefriel.it/mapproxy/MappointProxy.asmx

Figure 8: The binding list for second and third scenarios

P.	Composition	Activity name	Concrete service endpoint	Bind Date	St.
5	BusinessTrip_global	getCallInformation	http://163.162.173.70:8001/WsGateway/services/ppxSipTpc	2006-11-28 16:03:20	✓
5	BusinessTrip_global	requestMeeting	http://secse.cefriel.it:8084/axis/services/ServiceSoap	2006-11-28 16:04:04	✓
5	BusinessTrip_global	findDuration	http://secse.cefriel.it:8084/axis/services/xnavigationPort2	2006-11-28 16:02:41	✓
5	BusinessTrip_global	sendSms	http://secse.cefriel.it:8084/axis/services/SendSms2	2006-11-28 16:02:32	
1	BusinessTrip_global	sendSms	http://163.162.173.70:8001/WsGateway/services/ppxSmsSend	2006-11-28 16:04:15	✓
5	BusinessTrip_global	calculateTime	http://secse.cefriel.it:8084/axis/services/timeservicePort	2006-11-28 16:02:49	✓
5	BusinessTrip_global	getSmsDeliveryStatus	http://secse.cefriel.it:8084/axis/services/SendSms2	2006-11-28 16:02:32	
1	BusinessTrip_global	getSmsDeliveryStatus	http://163.162.173.70:8001/WsGateway/services/ppxSmsSend	2006-11-28 16:04:28	✓
5	BusinessTrip_global	checkSchedule	http://secse.cefriel.it:8084/axis/services/ServiceSoap	2006-11-28 16:02:57	✓
5	BusinessTrip_global	PointInPoly	http://secse.cefriel.it:8084/axis/services/MappointProxySoap	2006-11-28 16:02:32	
1	BusinessTrip_global	PointInPoly	http://server.democefriel.it/mapproxy/MappointProxy.asmx	2006-11-28 16:04:38	✓
5	BusinessTrip_global	makeCall	http://163.162.173.70:8001/WsGateway/services/ppxSipTpc	2006-11-28 16:03:05	✓
5	BusinessTrip_global	FindNearby	http://server.democefriel.it/poiservice/POIservice.asmx	2006-11-28 16:04:38	✓

Figure 9: The global binding list

From this experience we have learned new requirements that are being tackled in the second part of the project. In particular, we have had a confirmation of the importance of discovery mechanisms offered at various steps of the development and execution processes, and we have realized the need for searching not only based on functional properties of services, but also on their non-functional properties. Moreover, we have understood the importance of being able to access to various distributed registries for search.

In our example we had to tackle with different services for finding parking places that were defining different WSDL interfaces. To enable dynamic binding for these, we had to manually modify the code of the proxies automatically generated during the deployment phase. This has highlighted the need for studying the problem of adaptation to different WSDL interfaces more deeply.

The interaction with the telecom services offered by our end users has raised the issue of Service Level Agreement (SLA) definition, negotiation, and enforcement. We are analysing

the issue from the various aspects of the service life cycle, ranging from the definition of its interface to identify so called *SLA templates*, to the automatic support to the negotiation process, to the automatic generation of monitoring policies that can guarantee the enforcement of the agreement.

At last, another important aspect that has been, again, evidentiated by the interaction with the telecom services concerns the need for managing various approaches for communicating identities to the component services. In our example, again, we had to manually rework proxies to solve this issue. Currently, we are developing an identity management service that will abstract all other elements of a service-based system from this specific aspect.

5. ACKNOWLEDGMENTS

This work is funded by the European Commission VI Framework IP Project SeCSE (Service Centric System Engineering) (*http://secse.eng.it*), Contract No. 511680.

6. REFERENCES

[1] Service-Centric System Engineering. http://secse.eng.it.

[2] A4-D8 - Policies Specification and Integration with Existing Standards and Components. *SeCSE Deliverable*, 2006. http://secse.eng.it.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, K. J., F. Leymann, L. K., D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services. *http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/*.

[4] L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Proc. of the 3rd International Conference of Service-Oriented Computing (ICSOC 2005)*, volume 3826 of *Lecture Notes in Computer Science*, pages 269–282, Amsterdam, The Netherlands, 2005.

[5] L. Baresi and M. Miraz. A Distributed Approach for the Federation of Heterogeneous Registries. In *Proc. of 4th International Conference of Service-Oriented Computing (ICSOC 2006)*, pages 240–251, Chicago, IL, USA, 2006.

[6] M. Colombo, E. Di Nitto, M. Di Penta, D. Distante, and M. Zuccalà. Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems. In *Proc. of the 3rd International Conference of Service-Oriented Computing (ICSOC 2005)*, pages 48–60, Amsterdam, The Netherlands, 2005.

[7] M. Colombo, E. Di Nitto, and M. Mauri. SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *Proc. of the 4th International Conference of Service-Oriented Computing (ICSOC 2006)*, volume 4294 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2006.

[8] M. Di Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. Di Nitto. WS Binder: a framework to enable dynamic binding of composite web services. In *IWSOSE '06: Proc. of the 2006 international workshop on Service-oriented software engineering*, pages 74–80. ACM Press, 2006.

[9] J. Robertson and S. Robertson. Volere Requirements Specification Template. *http://www.volere.co.uk/*.

[10] J. Walkerdine, J. Hutchinson, P. Sawyer, G. Dobson, and V. Onditi. A Faceted Approach to Service Specification. In *Proc. of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007)*, Mauritius, CA, USA, 2007. IEEE Computer Society.

[11] K. Zachos, N. Maiden, X. Zhu, and S. Jones. Discovering Web Services to Specify More Complete System Requirements. In *Proc. of the 19th International Conference of Advanced Information Systems Engineering (CAiSE 2007)*, pages 142–157, Trondheim, Norway, 2007.

[12] A. Zisman and G. Spanoudakis. Uml-Based Service Discovery Framework. In *Proc. of 4th International Conference of Service-Oriented Computing (ICSOC 2006)*, pages 402–414, Chicago, IL, USA, 2006.