Requirements for Service Composition

Harald Meyer
Dominik Kuropka

Technische Berichte Nr. 11
des Hasso-Plattner-Instituts
für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik
an der Universität Potsdam

Nr. 11

Requirements for Service Composition

Harald Meyer
Dominik Kuropka

Potsdam 2005

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Die Reihe *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam* erscheint aperiodisch.

Herausgeber:	Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam
Redaktion:	Harald Meye, Dominik Kuropka r
Email:	harald.meyer; dominik.kuropka }@.hpi.uni-potsdam.de
Vertrieb:	Universitätsverlag Potsdam Postfach 60 15 53 14415 Potsdam Fon +49 (0) 331 977 4517 Fax +49 (0) 331 977 4625 e-mail: ubpub@uni-potsdam.de http://info.ub.uni-potsdam.de/verlag.htm
Druck:	allprintmedia gmbH Blomberger Weg 6a 13437 Berlin email: info@allprint-media.de

Heft 11 (2005)
ISBN 3-937786-81-3
ISSN 1613-5652

Requirements for Service Composition

Harald Meyer and Dominik Kuropka

November 1, 2005

Executive Summary

This document describes the requirements analysis of the service composition component. 33 requirements are identified and categorised. A requirement is either mandatory or optional. The categorisation distinguishes between general, functional and non-functional requirements. In general requirements automated service composition is justified. Functional requirements include requirements regarding:

- *Elements of Composition* defines the building blocks of composition.
- *Control Flow* defines the order of the elements of composition inside a composition.
- *Data Flow* defines how data is exchanged between the elements of composition.
- *Data Model* defines how data elements are described.
- *Quality of Service* defines how optimisations for compositions are described.

Non-functional requirements include the usage of WS-BPEL as the foundation for a service composition language, performance requirements, and the ability to flexibly react to unexpected events. Table 6 gives a summary of all identified requirements.

Contents

1 Introduction

To be successful, organisations cooperate on a global level. The key enabling cooperation is the integration of their software systems inside the organisation and beyond organisational borders. Service orientation proves to be a viable approach towards integration. Services are offered to requestors inside and outside the organisation. Descriptions of services are published in a registry. Service composition yields the possibility to aggregate existing services into new composed services. Processes are currently modelled manually. This leads to inflexible service compositions with no optimal quality as manual modelling prevents adjustments for individual service requests. Manually modelled service compositions must suit a lot of different service requests. Hence individual requests are possibly not served as well as possible. As services are registered and removed during run-time, service compositions may fail if necessary services are no longer available. When new services are registered they are not automatically used in previously modelled service compositions even though they may enhance them. Automated service composition allows the on-demand creation of new service compositions for requests. Therefore automated service composition increases flexibility and quality and reduces the probability of enactment failures because of missing services.

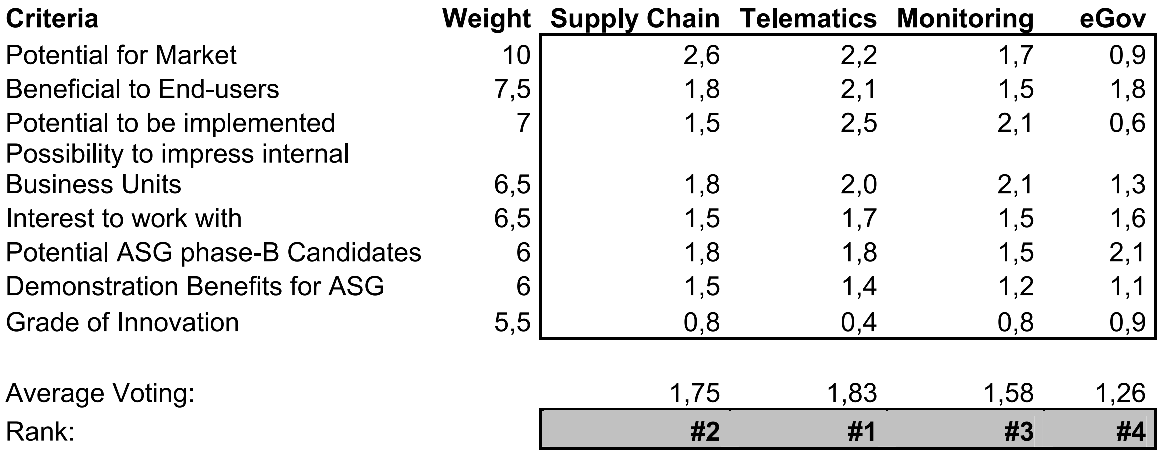
Automated service composition represents a core concept of the Adaptive Services Grid (ASG) project. One of the goals to achieve in the ASG project is to develop of a prototypical implementation of a service composition component. It will enable us to compose individual service compositions for service requests directed towards the ASG platform. Requirements analysis builds the foundation for the development of this component. Requirements analysis is part of the ASG platform development process described in the deliverable D6.IV-1 *ASG Platform Development Process*. It leads to a set of unambiguous, correct, and testable requirements for the service composition component. This includes functional requirements on the input and output of service composition but also non-functional requirements like performance.

The usage scenarios of the ASG platform developed by the working group *C-7: Usability and Demonstration* drive the requirements analysis. We will derive requirements from individual scenarios and demonstrate their importance through other scenarios and references to related work. Requirements analysis incorporate the collection and categorisation of requirements. Categorisation is performed on two dimensions: by importance and thematically. A requirement is either mandatory or optional. The thematical categorisation includes three main categories: general, functional and non-functional. General requirements are mainly requirements derived from the overall requirements regarding the ASG platform. This includes for example that service composition must be automated. While the general requirements can (and will) also be justified through the usage scenarios they precess the other requirements temporarily and logically.

Following this introduction two scenarios are presented in the next chapter that form the basis for our requirements analysis. General requirements regarding the service composition component are covered in chapter 3. In chapter 4 we present the functional requirements. We show the non-functional requirements in chapter 5. Chapter 6 contains a summary of all presented requirements and an outlook on design and implementation of the service composition component.

2 Use Case Scenarios

Two scenarios are used to motivate the individual requirements. Figure 1 shows an evaluation of selected scenarios. Based on this evaluation the selected scenarios are Telematics and Dynamic Supply Chain Management. From the Telematics scenario only a subset called Attraction Booking scenario is used here. It bases on the Tourist scenario presented in the deliverable D7-I.1 [13]. It plays an important role in the ASG project as it is implemented for the prototype of the ASG platform. The Dynamic Supply Chain scenario demonstrates service composition very well. Both scenarios are summarised in the following.



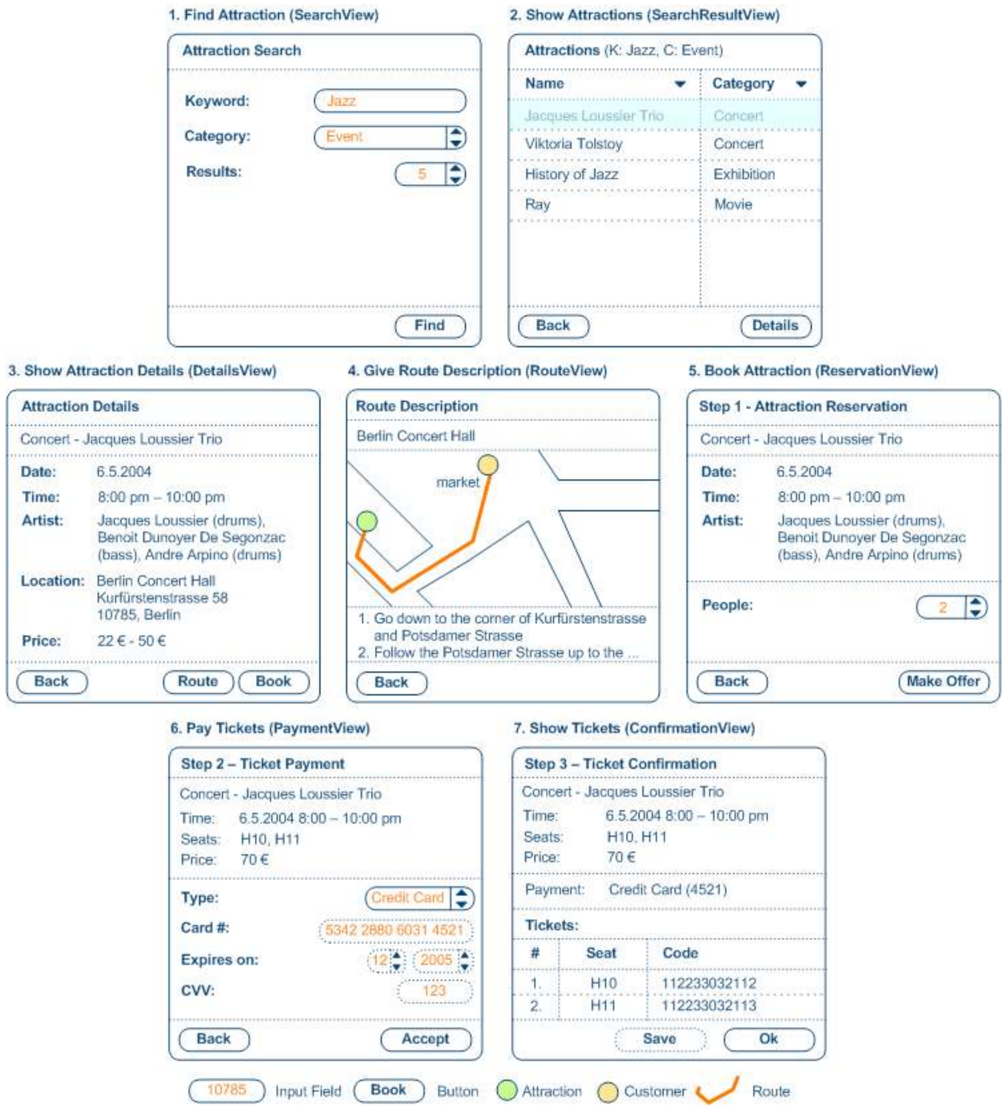
Criteria	**Weight**	**Supply Chain**	**Telematics**	**Monitoring**	**eGov**
Potential for Market	10	2,6	2,2	1,7	0,9
Beneficial to End-users	7,5	1,8	2,1	1,5	1,8
Potential to be implemented	7	1,5	2,5	2,1	0,6
Possibility to impress internal Business Units	6,5	1,8	2,0	2,1	1,3
Interest to work with	6,5	1,5	1,7	1,5	1,6
Potential ASG phase-B Candidates	6	1,8	1,8	1,5	2,1
Demonstration Benefits for ASG	6	1,5	1,4	1,2	1,1
Grade of Innovation	5,5	0,8	0,4	0,8	0,9
Average Voting:		1,75	1,83	1,58	1,26
Rank:		**#2**	**#1**	**#3**	**#4**

Figure 1: Evaluation of Scenarios [13]

2.1 Attraction Booking

The attraction booking scenario is a refinement of the Tourist scenario from the Telematics domain. Customers, or as they are called in ASG *end service consumers*, can retrieve information about attractions in the immediate surrounding using a mobile device. Additionally customers may request directions to the attraction or book tickets for them. Figure 2 shows the dialogs of the client application on the mobile device.

Based on the customer's current position and the specified query (dialog 1) the system compiles a list of attractions (dialog 2). This list is displayed as it is or as a map showing the position of the customer and the attractions found. The customer can then request additional information for specific locations. This includes the address of the attraction, opening hours or price (dialog 3). The customer can also request a route description to lead him to the attraction. If the attraction is bookable (e.g.: if it is an event), the customer can book a digital ticket. The route description consists of a map and a textual description (dialog 4). While the map shows streets and buildings, indicating



1. Find Attraction (SearchView)

Attraction Search

Keyword:	Jazz
Category:	Event
Results:	5

2. Show Attractions (SearchResultView)

Attractions (K: Jazz, C: Event)

Name	Category
Jacques Loussier Trio	Concert
Viktoria Tolstoy	Concert
History of Jazz	Exhibition
Ray	Movie

3. Show Attraction Details (DetailsView)

Attraction Details

Concert - Jacques Loussier Trio

Date:	6.5.2004
Time:	8:00 pm – 10:00 pm
Artist:	Jacques Loussier (drums), Benoit Dunoyer De Segonzac (bass), Andre Arpino (drums)
Location:	Berlin Concert Hall Kurfürstenstrasse 58 10785, Berlin
Price:	22 € - 50 €

4. Give Route Description (RouteView)

Route Description

Berlin Concert Hall

1. Go down to the corner of Kurfürstenstrasse and Potsdamer Strasse
2. Follow the Potsdamer Strasse up to the ...

5. Book Attraction (ReservationView)

Step 1 - Attraction Reservation

Concert - Jacques Loussier Trio

Date:	6.5.2004
Time:	8:00 pm – 10:00 pm
Artist:	Jacques Loussier (drums), Benoit Dunoyer De Segonzac (bass), Andre Arpino (drums)
People:	2

6. Pay Tickets (PaymentView)

Step 2 – Ticket Payment

Concert - Jacques Loussier Trio

Time:	6.5.2004 8:00 – 10:00 pm
Seats:	H10, H11
Price:	70 €
Type:	Credit Card
Card #:	5342 2880 6031 4521
Expires on:	12 2005
CVV:	123

7. Show Tickets (ConfirmationView)

Step 3 – Ticket Confirmation

Concert - Jacques Loussier Trio

Time:	6.5.2004 8:00 – 10:00 pm
Seats:	H10, H11
Price:	70 €
Payment:	Credit Card (4521)

Tickets:

#	Seat	Code
1.	H10	112233032112
2.	H11	112233032113

Figure 2: Attraction Booking: client application

the path as a coloured line, the textual description includes street names and directions (e.g.: where to turn left or right). To book a specific attraction or event, the customer selects the number of people to attend the event (dialog 5). The system then offers the customer a number of tickets for a certain price category. If the customer decides to accept the offer, he is charged the specified amount of money (dialog 6). In return he receives the given number of digital tickets for the event (dialog 7). The Attraction Booking scenario encompasses three use cases:

- Find attractions
- Get route description
- Book attraction

All three are fulfilled through the enactment of service compositions. Figure 3 shows an example composition for each use case. The process activities represent service invocations. The service compositions result from service requests towards the service composition component. For example, if the customer's location is already specified in the service request, the invocation of a localisation service is not necessary. A similar flexibility applies to the booking of an attraction. No payment option is specified in the example. Instead the user selects a payment option during payment. But the customer may have already specified his preferred payment option in a profile. The client application on the mobile device then transfers the information about the preference and the customer does not have to select a payment option. If service providers register new attraction information services or remove old services, future queries for attraction information will incorporate these changes of the service landscape. In this scenario automated service composition allows changes in the set of available services and in the client application (new use cases, additional information) without requiring manual adjustment of the service compositions.

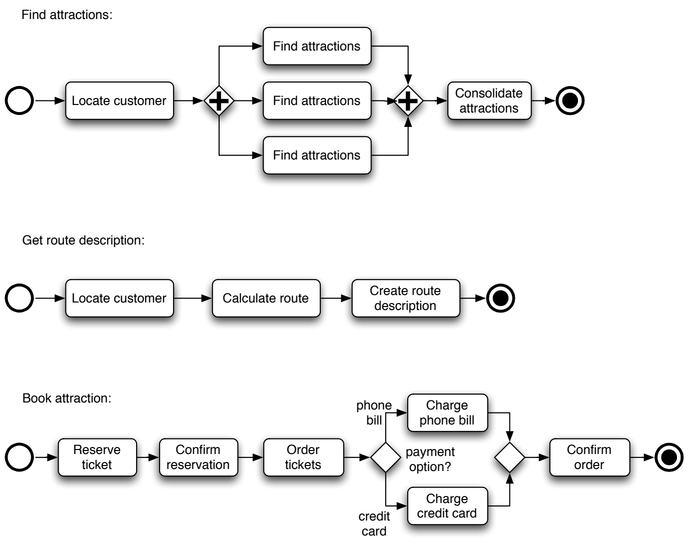


Figure 3: Example compositions for attraction booking use cases

2.2 Dynamic Supply Chain Management

To be successful, companies must be able to integrate services offered by their suppliers quickly. In the Dynamic Supply Chain Management scenario this integration should be achieved for the domain of Internet Service Providers (ISP).

Nowadays ISPs not only provide broadband Internet access but bundle it with additional services like webspace provision or domain name registration. Bundling can be done in advance by a product manager or on-demand when the customer requests a product bundle. Bundled services may not be provided by the ISP itself but by one of its suppliers: While the ISP owns the infrastructure for providing access to the Internet, it purchases the actual broadband lines connecting customers (e.g.: ADSL line over a regular telephone line) from line carriers.

If a customer orders a bundled package, the online shop of the ISP creates a service request that is send to the ASG platform. Based on this request a fitting service composition is created. The use case for this scenario is ordering of a broadband Internet access bundle. Figure 4 displays a composition example for this use case. The customer requests a bundle containing broadband Internet access, web space and the registration of domain names. Service enactment can enact all three activities in parallel. To provide Internet access the ISP not only orders a ADSL line from a carrier, but also performs setup activities for its internal infrastructure (e.g.: account to log in). While ordering an ADSL line and registering a domain name are displayed as activities, they are actually compositions themselves. Figure 5 shows the actual service invocations for ordering an ADSL line. An availability check is performed for each ADSL provider. The provider is added to the list of possible providers, if the check returns a positive result. When the provider check is complete, the best offer (e.g.: least cost) is selected.

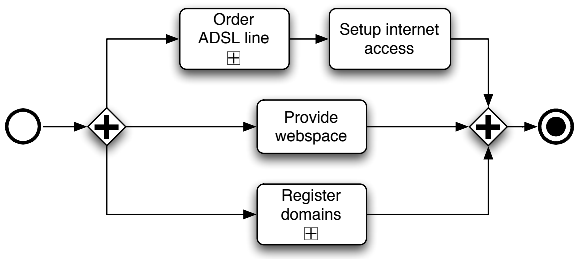


Figure 4: Example composition for Dynamic Supply Chain Management

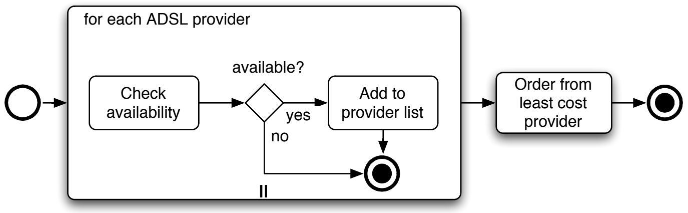


Figure 5: Sub process ADSL line ordering

Automated service composition allows customers to select individual bundles including only the desired products. The service composition component can flexibly compose the ordering service of the ISP according to the requested bundle. Automated service composition also enables the ISP to have business relations with a lot of different suppliers. The best provider for a concrete product is then selected dynamically. New business relations can therefore be established on the fly and old ones can end.

3 General Composition Requirements

This chapter gives an overview of general requirements towards service composition in ASG.

Req. 1: Service Composition is automated (mandatory) Service composition can be performed manually, semi-automatically, or automatically. Systems following the paradigm of service orientation are open: Service requestors appear and disappear, and service providers register new services, and change or unregister existing services. Service requestors are the customers of the end service provider. In the Attraction Booking scenario the customer is somebody who wants to find and book an attraction. In the Dynamic Supply Chain Management scenario the customer wants to order bundled ISP services. The end service provider is the organisation that offers the actual service to this customer. He provides the client application, the server application and often also the ASG platform. End service providers may change their applications. New customers may have new demands. Handling these demands with manually modelled compositions can be inadequate or impossible. In Dynamic Supply Chain Management a lot of different services are available that can be bundled into even more packages[1]. Manually modelled service compositions that handle all this different cases are complex. Complexity increases, if dependencies between services exist (e.g.: domain registration is only possible if web space is ordered, too) or if providers register or unregister services.

Service providers register their services and offer them to customers. A provider of a route planning service is an example for a service provider in the Attraction Booking scenario. For Dynamic Supply Chain Management an example is the provider of ADSL lines. In both cases providers can register new services. This includes not only new route planning services or ADSL line ordering services but also new types of services. A provider of e-mail accounts may want to offer its services in the Dynamic Supply Chain Management. With manual or semi-automatic service composition the service provider asks the end service provider to adjust its service composition in order to allow customers to include an e-mail account in their package. If the end service provider uses automated service composition, new services will be used automatically. A service provider may also change or unregister its services. Again this will result in manual modelling if no automated service composition is used.

Finally, the end service provider can change its application. Using automated service composition, he only needs to change the service requests. The service composition component automatically adjusts the service compositions. If the provider decides that the information about an attraction should include a live picture of the location, he only changes the goals of the service request. Without automated service composition, he would have to change the service compositions himself.

Each situation presented above involves some change in the service compositions. For manual or semi-automatic service composition this means that a person has to change the compositions manually. This is time-consuming and costly. With automated service composition no service compositions to change exist. Adaptation to changes resulting from the above mentioned openness of

[1]If five service are available and a bundle may include each service once, 32 packages are possible.

service oriented architectures is therefore easier.

Req. 2: Service request describes the goal of composition (mandatory) A service request is the input for automated service composition. If service enactment executes the resulting service composition, its effect should be in line with the service request.

To allow automated service composition a service request includes initial state, goal state and request data. The initial state describes the current situation before enacting the service composition. The goal state describes the state of the world that should be reached. Request data includes concrete data elements that are available in the initial state. In the Attraction Booking scenario this includes for example the customer's telephone number.

Req. 3: Service Composition according to service request (mandatory) The service composition component creates compositions, that accord the service request: The service composition is enactable in the initial state with the given request data and enacting the service composition leads to a state that fulfils the criteria defined for the goal state.

Req. 4: Information gathering services invocable during composition (optional) One can distinguish information-gathering and world-altering services [9]. Information-gathering services only fetch new information without changing the state of the world. In the Attraction Booking scenario the services to fetch information about attractions and to locate customers are only information gathering[2]. In contrast, ticket booking leads to a change in the state of the world: Fewer tickets are available and money is transferred. Therefore the booking service is world-altering.

In the classical view of automated service composition neither information-gathering nor world-altering services are called during service composition. Instead the service composition component puts the selected service into the service composition. The service is therefore only invoked during service enactment, While this separation of concerns normally makes sense, it can lead to bloated service compositions. If a search for attractions is performed in the Attraction Booking scenario, all attraction information services regardless of the location of their attractions are used. This happens because the location is determined during enactment. If the service for locating the customer could be invoked during composition time, the number of applicable attraction information services is reduced dramatically.

While some research on the enactment of information-gathering services during composition time exists [9] this interweaving of composition and enactment can be problematic: How should the service composition component decide whether to invoke a service itself or to let the service enactment component do this? Is an information-gathering service that costs money still information-gathering? Another problem is that only very few information gathering services are actually invocable during composition time as they depend on world-altering services [17].

[2]Only if they do not cost any money.

4 Functional Requirements of Service Composition

In this chapter we elaborate the functional requirements towards the service composition component and the composition language. The composition language defines how service compositions in ASG look like. The requirements are ordered according to the following categorisation:

- *Elements of Composition* defines the building blocks of composition.
- *Control Flow* defines the order of the elements of composition inside a composition.
- *Data Flow* defines how data is exchanged between the elements of composition.
- *Data Model* defines how data elements are described.
- *Quality of Service* defines how optimisations for compositions are described.

Following this model the requirements are described in the following.

4.1 Requirements on the Elements of Composition

The elements of composition are the building blocks from which service compositions are composed. The service specification language defines their concrete format. Requirements toward the service specification language are defined the deliverable D1.I-1 *Requirements Analysis on the ASG Service Specification Language* [14].

Req. 5: Elements of composition are services interactions (mandatory) The elements of a composition are activities that perform a task. In ASG the only activities are service interactions. Besides invoking services, a composition can itself be invoked as a service. The result of composition includes service interactions only on the specification level without a binding to a concrete implementation. The Negotiation Manager later binds the specifications to concrete implementations. If an external service provides more than one method, each method is mapped to a distinct ASG service. Therefore no stateful interactions with services containing more than one method are supported.

For the scenarios this means that every activity must be invocable as a service. This restriction includes all internal activities. For example in the Dynamic Supply Chain Management scenario the activities to create an account and to reserve disk space on a web server are services, too.

Req. 6: Elements of composition are service compositions (optional) Service interactions are the atomic elements of composition. To improve reusability existing service compositions are used as elements of composition as well. These existing service compositions can be manually modelled or stored results of previous automated service composition.

Using manually modelled composition certain sequences of services can be prescribed to the composer. In the Attraction Booking scenario such a predefined fragment could be the charging process. Instead of recomposing it for each request, it could be predefined dictating a certain order in which the different charging services are used (e.g.: charging by money transfer only if no other payment options succeeds.). Reusing previous automated service compositions makes sense in most scenarios especially if often similar or equal requests are served. The amount of automated service composition is reduced and reply times are decreased.

Req. 7: Services have input and output parameters (mandatory) Services perform a specified functionality. Often this functionality is parameterised with data provided in the service request. The localisation of a customer for example needs the customer's telephone number as input data. Services also return a result (e.g.: the customer's location). To allow the input and output of data, parameters are needed. A service has zero or more input and output parameters.

Req. 8: Service functionality is described semantically (mandatory) Service functionality is described semantically to allow automated service composition. Besides input and output parameters, specifications of services include preconditions and effects. These four elements of a semantic service specification can be mapped to the WSMO constructs of assumption, precondition, effect, and post-condition. The terminology is derived from automated planning in Artifical Intelligence [7] and is in line with the current ASG terminology. Preconditions and effects are specified using function-free first-order logic. This allows semantically rich specifications while preserving decidability [7]. Preconditions and effects describe the state of the world and the state of available information. To do so, it is possible to define logical relations between input parameters, between output parameters and between input and output parameters. To describe functionality often additional variables that are not parameters are necessary.

Req. 9: Services can have more than one precondition or effect (mandatory) It is common that a service has more than one precondition or effect. For example the service to provide web space in the Dynamic Supply Chain Management scenario has the preconditions that enough disk space is available and that the load on the machine is acceptable. It has the effects, that web space is reserved and that an account for the user is created. All of the preconditions must hold in order to execute the service and all effects of the service happen if the service is executed.

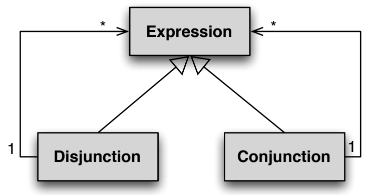


Figure 6: Composition of logical expressions

Expressions are defined in first-order logic. Therefore expressions can be conjunctions or disjunctions of other expressions (Fig. 6). Conceptually services have only one precondition and one effect. If more than one precondition or effect is needed, both can be conjunctions of other expressions. The logical expression for the precondition of the above-mentioned service could look like this:

$$spaceAvailable \wedge loadAcceptable$$

Req. 10: Services can have disjunctive preconditions (mandatory) Besides having multiple preconditions that all must be true in order to invoke the service, a service can have disjunctive preconditions. The service is invocable, if just one of the preconditions is true. An actual web space provisioning service could be responsible for multiple web servers. So it is sufficient that web space is available on just one web server. The precondition of the service managing web servers A and B then looks like this:

$$(spaceAvailable(A) \wedge loadAcceptable(A)) \vee (spaceAvailable(B) \wedge loadAcceptable(B))$$

Again, we can use just one expression from first-order logic to express disjunction. It is therefore sufficient that we still limit services to just one precondition. If disjunctive preconditions are not available, they can be simulated using several distinct services (e.g.: service for web server A, service for web server B).

Req. 11: Services can have disjunctive effects (mandatory) Disjunctive effects are necessary to express services that can have several different effects. The web space provisioning service that is responsible for multiple web servers is an example for such a service. Depending on the selected web server, the web space will be provided on just one of the managed web servers. In the Attraction Booking scenario a service to determine the city in which a customer is located delivers different results according to the actual city.

Implementation-wise disjunctive effects are more complicated. While missing disjunctive preconditions can be simulated, this is not possible for disjunctive effects. Disjunctive effects also increase the complexity of automated service composition [5, 7].

4.2 Control Flow Requirements

The control flow of a process or service compositions defines the order in which the elements of composition are enacted. This includes simple sequential ordering but also complex parallel or alternative control flows. In the usage scenarios we have already seen some examples for control flows. Figure 3 for the Attraction Booking scenario and Figures 4 and 5 for the Dynamic Supply Chain Management scenario included sequential, parallel, and alternative control flows.

Requirements regarding control flow can be separated into two types of requirements: Requirements regarding the composition functionality and requirements regarding features of the composition language. With workflow patterns [18] a categorisation for different control flow constructs exists in workflow management. Requirements analysis regarding control flow will be performed according to these patterns.

Req. 12: Composition of sequential control flow (mandatory) In a *Sequence* of activities the activities are enacted one after another in a well-defined order. Figure 7 shows examples for the usage of *Sequence* in both scenarios. In the Attraction Booking scenario the customer is located, then a route is calculated based on the location and finally a description for the route is generated. In the Dynamic Supply Chain Management example only a small part of the process is actually performed sequential. If an ADSL line is ordered by a customer, the actual line is ordered from the ADSL line provider and then Internet access is activated for this line. In both scenarios sequentialisation is necessary to ensure correct working.

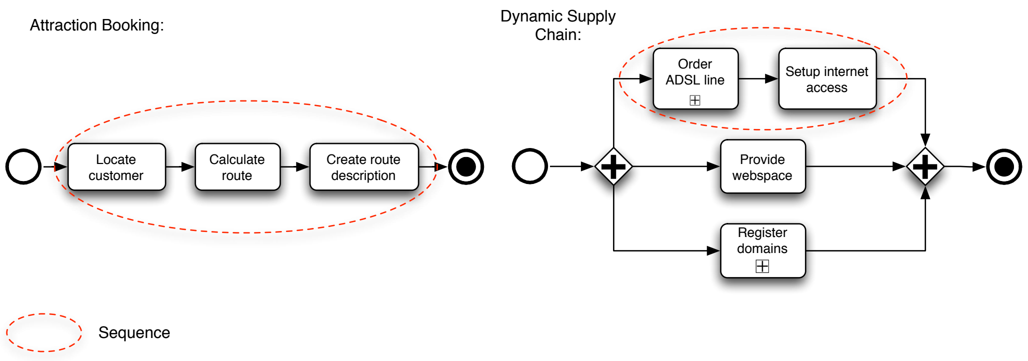


Figure 7: Usage of Workflow Pattern: Sequence

Req. 13: Composition of parallel control flow (mandatory) Parallel control flow allows the parallel invocation of activities. We have seen an example for parallel control flow in the Dynamic Supply Chain Management scenario in Figure 7. It is also used in the Attraction Booking scenario. In general every usage of parallel control flows can be sequentialised. Instead of executing ADSL line ordering, web space provisioning and domain registration in parallel, they could be executed in sequence. But sequentialisation can dramatically reduce process performance. For the application to real world scenarios it is therefore mandatory.

Parallel control flow is realised by two different patterns: *Parallel Split* and *Synchronization*. A Parallel Split splits a single thread of control into multiple threads. A Synchronization merges them later. Figure 8 identifies both patterns in an example from Attraction Booking.

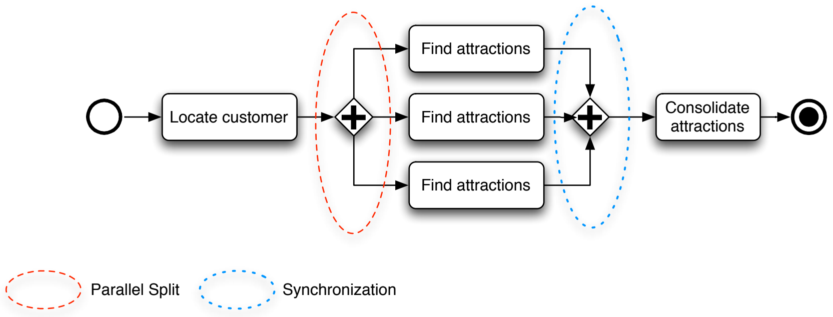


Figure 8: Usage of Workflow Patterns: Parallel Split and Synchronization

Req. 14: Composition of alternative control flow (mandatory) Alternative control flows are parts in a process where – depending on some condition – one out of many possible control flows is selected. One example from the Attraction Booking scenario is displayed in Figure 9. It shows the realisation of an alternative control flow using the workflow patterns *Exclusive Choice* and *Simple Merge*. Depending on the preferred payment type, either the customer's phone bill or his credit card is charged. Conditions are often either decided by user provided data (like here) or by services with disjunctive effects.

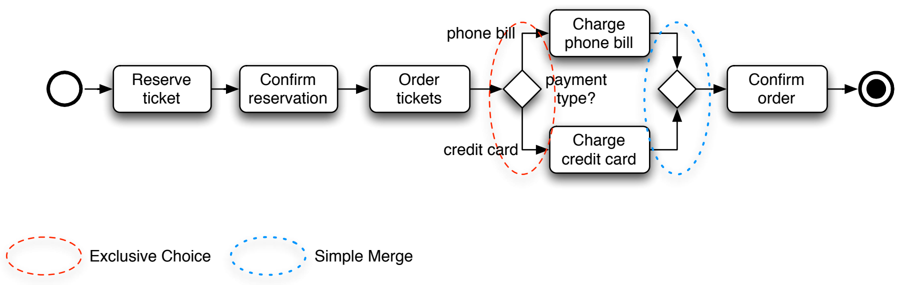


Figure 9: Usage of Workflow Patterns: Exclusive Choice and Simple Merge

The patterns *Exclusive Choice* and *Simple Merge* constitute the simplest form of alternative control flow: Exactly one of the alternative control flows is selected. If a *Multiple Choice* is used instead, multiple alternative flows can be taken. To merge them three different patterns *Synchronizing Merge*, *Discriminator* and *Multiple Merge* can be used. Only *Synchronizing Merge* performs synchronisation of control flows. In contrast to *Multiple Merge*, the *Discriminator* pattern executes subsequent activities only once.

In general the service composition component should support at least *Multiple Choice* as a splitting pattern. It can simulate *Parallel Split* and *Exclusive Choice* as special cases. Merging patterns are more complex. Of the synchronising pattern – *Synchronization*, *Simple Merge* and *Synchronizing Merge* – only the last one is necessary. Non-synchronising patterns are currently not possible with automated service composition algorithms.

Req. 15: Composition of multiple instances of sub-compositions (optional) Sometimes it is necessary to execute certain tasks not only once, but multiple times. ADSL line ordering in the Dynamic Supply Chain Management scenario is one example (see Figure 5). Several providers for ADSL lines exist. They differ in availability and price. So an availability check for each provider must be performed and the cheapest one is selected. The availability check for each provider is a realisation of one multiple instance pattern: *Multiple Instances with a priori known design time knowledge*. Of the four different multiple instance pattern it is the only one that can be composed automatically. As the number of providers is already known at design time (here: composition time) parallel branches for every provider are created (Figure 10).

Other multiple instance workflow pattern capture situations were the number of instances is known only at runtime or not known at all. Composing this automatically is not possible. Nevertheless, situations where the number of instances is known at run time can be handled through the Negotiation manager of ASG.

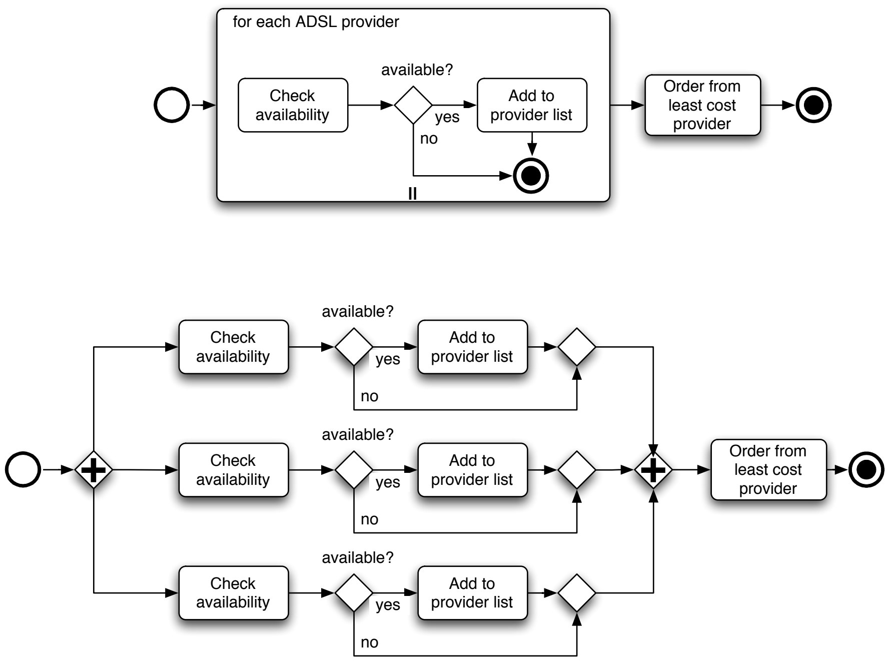


Figure 10: Multiple Instance pattern example and its composition

Req. 16: Composition Language supports workflow patterns (mandatory) So far all control flow requirements were requirements regarding the functionality of the composition component. The actual output as an instance of the composition language is important as well.

The basic requirement on the composition language regarding control flow is the support of the above-mentioned required workflow pattern. This can be achieved through a graph-structured or a block-structured approach. In a graph-structured approach activities are vertices that are connected through edges that symbolise ordering constraints. In a block-structured approach structured activities exist that contain other activities and determine their enactment order. While a graph-structured approach is more generic, is a block-structured approach easier to visualise and reason about. Reasoning on process structures is for example necessary during negotiation to adhere to quality of service properties. In general it is best to support both approaches like WS-BPEL [12] does. The composition language should also support the patterns that cannot be composed automatically. This makes sense as automated composition can reuse manually modelled process fragments.

Req. 17: Composition is block-structured (optional) Supporting structured activities in the composition language does not mean that the composer outputs a block-structured result. Actually, service composition generates a partially-ordered set of services that can be represented as a graph. As already mentioned block-structuring has its advantages and should be preferred when possible. To have block-structured compositions, either the composition algorithm could support them directly or post-processing could be performed. The first approach is a new research area, so it is unclear whether it can be successful. Hierarchical-Task-Network planners [15, 4, 11] generate block-structured plans. But they do not actually generate block structures, but rather reuse them. Post-processing rises the problem of complexity. As shown in [2] reordering compositions subsequently can be as complex as composition itself.

4.3 Data Flow Requirements

The data flow of composition defines how data is exchanged between the service. Services have input and output parameters. Output parameters of one service can be the input for other services. Data flow requirements include for example the ability to exchange data and the usage of process input data. The following four requirements regarding data flow are all defined over activities instead of services. This is valid as the activities represent invocations of services and the actual data exchange is done between the activities.

Req. 18: Activities exchange data (mandatory) The fundamental data flow requirement is the ability to exchange data. Exchanging data between activities and therefore data flow is supported in nearly all process meta-models [3, 10]. Activities have formal parameters that are replaced by actual data when invoked. This data can either be process input data or output from other activities.

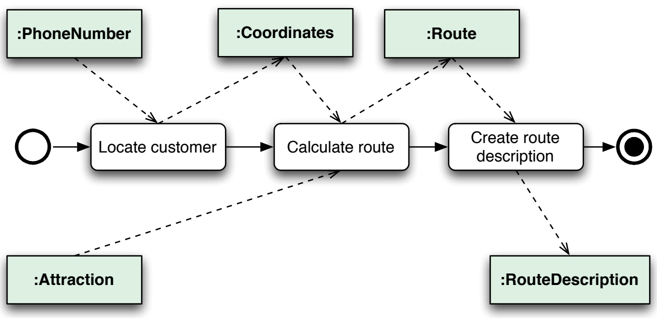


Figure 11: Data flow in the Attraction Booking Scenario

Data flow is used in both use case scenarios. Figure 11 illustrates it for one composition from the Attraction Booking scenario. The green rectangles are data elements that are exchanged between the activities. An arrow leading to a data element means that it is created by the originating activity. The customer coordinates are for example created by the customer localisation service. An arrow leading from a data element to an activity symbolises that this data element is an input parameter for the activity. For example, the coordinates are input to the route calculation service.

In workflow management, two different approaches to model data flow are in use. With the first approach all data is stored on the process level. Input parameters of activities are read from this central storage. Output parameters are stored in this central storage or – as it is called – *blackboard*. With the second approach, data actually flows between activities. Explicit data flow connectors connect the outputs of one activity with the input of another one. So the main difference is that in the first approach all data exchange must be done through the central storage. If the output of one activity is used by two other activities, it is still written only once to the storage and read twice. In the second approach two distinct data connectors exist. WS-BPEL uses the blackboard approach through process variables [12]. In contrast, Leymann and Roller [10] propose a meta-model, used in IBM MQSeries Workflow, that facilitates explicit data connectors. The flexibility gained by the blackboard approach, stands in contrast to its harder to follow – implicit – data flow. This requirement and service composition in general are agnostic to the actual approach selected.

Req. 19: Activities use process input data (mandatory) Figure 11 shows the data flow for an example composition of the Attraction Booking Service. Certain data elements, like *PhoneNumber* and *Attraction*, are not produced by any activity. These data elements are part of the process data and are inputs for the process. Processes must have such data, and activities must be able to use them.

Req. 20: Data exchange implies control flow (mandatory) Control flow embeds an ordering constraint between two activities if one activity depends on another one. Dependencies are for example causal links (e.g.: an activities creates the precondition of another one) or the protection of causal links. Causal links do not only exist on the level of semantic service descriptions, but also for input and output parameters. If one activity uses the output of another activity as an input a causal link between the two activities exist. Therefore an ordering constraint between the two activities must be included.

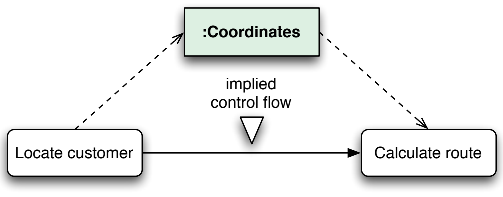


Figure 12: Control flow implied by data flow

For example see Figure 12: Even if the customer localisation service and the route calculation service are not linked through preconditions and effects, it is necessary to add an ordering constraint between them as the route calculation service uses the coordinates created by the localisation service.

Req. 21: Activities create new variables (mandatory) While this requirement sounds trivial and self-evident, it actually is not for automated planning. Activities create new variables, means that activities do not just write data into already defined variables, but they create variables on the fly. With automated planning this is normally not possible. All the variables that are usable during composition must be defined in advance. This includes also intermediate variables that are neither used in the input nor in the output. When retrieving a route description in the Attraction Booking scenario, a coordinates variable must be defined. This coordinates variable is never used in the input or the output. It is also not obvious why such a variable could be necessary. Hence, by adding this variable we are encoding assumptions about automatically created service composition into the service request. This is bad as it hampers flexibility. Other service compositions are possible that do not need this variable. Figure 13 shows such an example. If a combined localisation and route calculation service exists, the customer's coordinates are not necessary.

Defining all necessary variables requires a lot of information about the available services and at least a rough idea on how the composition could look like. Therefore it is required here that activities can create new variables and that the service composer takes these into account.

Req. 22: Scoping of process data (optional) Scoping introduces the concept of local variables. The advantage is that sub-processes can define their own variables without fearing to clash

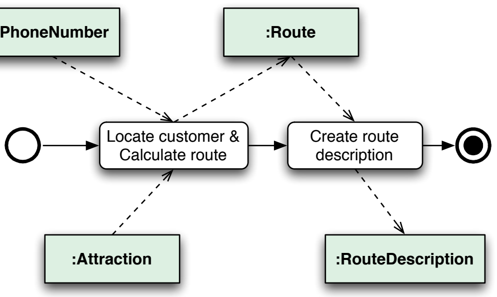


Figure 13: Data flow using combined localisation and route calculation service

with variables defined in other sub-processes or in its super-process. As mentioned, the output of composition is normally is partially ordered set of service invocations. As such a service composition is *flat*, scoping is not required. If service composition composes structured activities, scoping of process data must be supported.

4.4 Data Model Requirements

The data model defines how data elements are described. The data model is of importance for the service composition component, as it has to use data elements to replace the formal parameters with actual parameters.

Req. 23: Data elements are typed (mandatory) Data elements are exchanged between services as parameters. To ensure that only valid data elements are passed as parameters it makes sense to type them. Besides predefined types, user-defined types are necessary as well. In the previous section we have seen that in the Attraction Booking scenario types like *PhoneNumber*, *Coordinates*, and *Route* exist. By typing parameters we know for example what the localisation service requires as an input: Neither a string nor a number, but a phone number.

But type-safety not only prevents service enactment from invoking services with wrong parameters, it also eases planning. In the initial state of the *Get Route Description* use case of the Attraction Booking scenario a phone number and an attraction is known. With typed data elements and parameters the service composer knows that it can only use the phone number as a parameter for the localisation service. So the composer can already eliminate half of the potential compositions. Later, when more data is available this reduction is even more drastical. Optionally if types are not needed a generic type (e.g.: *Object*) can be used.

Req. 24: Data element types are defined in an ontology (mandatory) Service specifications are annotated semantically to allow automated service composition. Service specifications therefore include preconditions and effects. Both are modelled as logical expressions. To use input parameters, output parameters and variables in these logical expressions, the types of data elements are described in an ontology.

An ontology defines concepts and their relations. Figure 14 shows the ontology of the Attraction Booking scenario modelled using the Unified Modelling Language. A concept can be a sub-concept

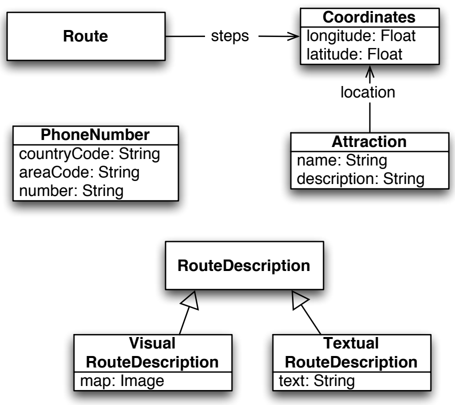


Figure 14: Extract from the Attraction Booking ontology

of another concept (*inheritance*). Aggregation and composition relations can also exist between concepts. As the example shows all these different relations are necessary.

As no mediation is supported, all concepts used in one scenario have to be defined in one ontology. A service provide is therefore required to use only these concepts to describe his services.

Req. 25: Composer is aware of data element structure (mandatory) Besides having data elements with ontology-based types it is also necessary that the composer is aware of the internal structure of data elements described using an ontology. In the Attraction Booking scenario the composer must be aware of the fact that an attraction has a coordinate. Only then it is possible to express that the use case *Get Route Description* delivers a route *to* the attraction.

To do so the composer has to not only understand function-free first-order logic but also Frame Logic [8]. Frame logic is an enhancement of first-order logic as it adds object-oriented concepts like object identity, inheritance and complex objects. An expression to model that we want to have a route that leads to the location of a given attraction looks like the following in Frame Logic:

$$Attraction[location \rightarrow ALoc] \wedge Route[endCoordinates \rightarrow REnd] \wedge ALoc = REnd$$

Req. 26: Data elements can be used to evaluate control flow conditions (mandatory) Requirement 13 above stated that the composer can compose alternative control flows. An alternative control flow can be the result of an *Exclusive Choice* or a *Multiple Choice*. Both have one incoming and multiple outgoing control threads. To decide which control threads are actually executed, conditions are assigned to the individual threads. In the Attraction Booking scenario alternative control flows are used to determine the payment type. In the Dynamic Supply Chain Management alternative control flows are necessary when the list of ADSL providers is generated. Based on the result of the availability check the provider is added to the list or not.

In both cases conditions are necessary. In the Attraction Booking scenario it is the payment type. In Dynamic Supply Chain Management the result of the availability check. The condition is expressed over data elements. So the composer must be able to model such conditions.

Req. 27: External application data exists (optional) The data we have seen in both scenarios until now always was structured and rather small. In other scenarios unstructured, huge amounts of binary data are exchanged between services. One example is the E-Government scenario introduced in the deliverable D7-I.1 [13]. The goal of this scenario is to give people access to information about legislation in progress. The law texts can be rather large documents stored in a central repository. Such documents tend to be several megabytes of data large. Often it does not make sense to model this data as implied by the previous requirements. Instead of transferring them as message parts, it makes more sense to just send references to them and tread their internals as a black box [1].

The implication of having such application data is that this data cannot be used as control flow data to evaluate conditions at for example a *Multiple Choice*. The service composer can also only handle the document through the reference. It cannot look at its internal structure and reuse parts of it. The support for external application data is an optional requirement as it is not necessary in the scenarios. In exchange for reduced performance, it is in most cases possible to work without external application data even if large documents are transferred.

4.5 Quality Of Service Requirements

The quality of service (QOS) requirements here are part of the functional requirements. They describe functionality that is needed to ensure quality of service properties for service compositions defined in a service request. The requirements on the representation of quality of service properties for service requests and service specifications is out of scope of this requirements analysis. They were already defined in *Requirements Analysis on the ASG Service Specification Language* [14].

Req. 28: Composition uses static quality of service properties (optional) Services have static and dynamic quality of service parameters. Static QOS properties define ranges or enumerations of possible values. The actual values are negotiated during run-time by the *Negotiation Manager*. For example, ordering an ADSL line in the Dynamic Supply Chain Management scenario costs a setup fee of 10 to 20 €. The actual value depends on the selected provider and on negotiated service level agreements.

As negotiation takes place after service composition, the negotiated values cannot be used during service composition. Hence, only static values can be used. Despite their fixed values, it makes sense to use them. The service composer can select service with equivalent functionality based on quality of service properties. Negotiation gets the best prerequisites to negotiate optimal service level agreements. This requirement is optional as the process of composition, negotiation and enactment works without it, too.

Req. 29: Composition fulfils quality of service properties from service request (optional) To use the quality of service properties to select the best service compositions, it is important to describe the relevant quality of service properties and their desired values in the service request. A service request in the Attraction Booking scenario could state that finding attractions costs at most 0.10 €. Accordingly, all services invoked can only cost 0.10 €. Then the service composer selects and composes service compositions according to these quality of service parameters.

If the composer is able to find a composition that cost a maximum of 0.05 €, it is ensured that the Negotiation Manager is able to negotiate a valid service level agreement. If costs range between

0.05 €and 0.15 €, the composer cannot ensure that negotiation is successful. Depending on negotiation skills and external events (e.g.: high load on service, that leads to higher prices), no valid service level agreement may be negotiable. Finally if minimum costs are 0.15 €negotiation is unnecessary as it will fail.

Therefore it makes sense for ASG platform that the service composer tries to fulfil quality of service properties using static values, as we can decide after composition whether negotiation makes sense.

5 Non-Functional Requirements

In the previous section the functional requirements for the service composer have been elaborated. Non-functional requirements exist as well. These non-functional requirements are only non-functional requirements regarding the service composer and not non-functional properties of the composition result.

Req. 30: Service composition language bases on WS-BPEL (mandatory) WS-BPEL [12] is a language for the modelling of web service compositions. WS-BPEL processes can be enacted automatically using a workflow engine. While WS-BPEL originated like most other WS-* standards from industry efforts, recently a technical committee was founded at the OASIS to standardise WS-BPEL.

As one of the goals of the ASG project is the use and enhancement of existing standards it is sensible to use a standardised language as the composition language. A further advantage is the wide-spread availability of workflow engines supporting WS-BPEL. This includes not only open source developments but also mature products. To strengthen the interoperability and reusability of the service composer, WS-BPEL should be supported.

Req. 31: Services are specified using the ASG Service Specification Language (mandatory) In the deliverables D1.1-1 *Requirements Analysis on the ASG Service Specification Language* requirements regarding the service specification language in ASG were captured. These requirements are the basis for deliverable D1.1-3 where the service specification language will be elaborated. The service composer will use service specifications in this language to perform automated service composition.

Req. 32: Service composition time must not exceed advantages from automation (mandatory) Automated service composition has the advantage of gained flexibility and potentially better service compositions. Service compositions are better according to quality of service properties like cost and time. Automating service composition only makes sense if the cost for automation are justified by its advantages.

Figure 15 displays three situations. In the first situation a manually modelled service composition is used. Compared to the two other situations enactment time is rather high. But as the composition is modelled manually in advance, no additional composition time is needed unless changes (e.g.: available services) occur. In the two other situations automatically composed service compositions are used. The compositions take less enactment time, as an adjustment to the exact service request has been performed. But despite lower enactment time, automated service composition only makes sense in the situation displayed in the middle. Combined composition and enactment time is less than the original enactment time. In the last situation displayed, the combined duration is higher than the original enactment time. Here automated service composition does not give advantages for the quality of service property execution time.

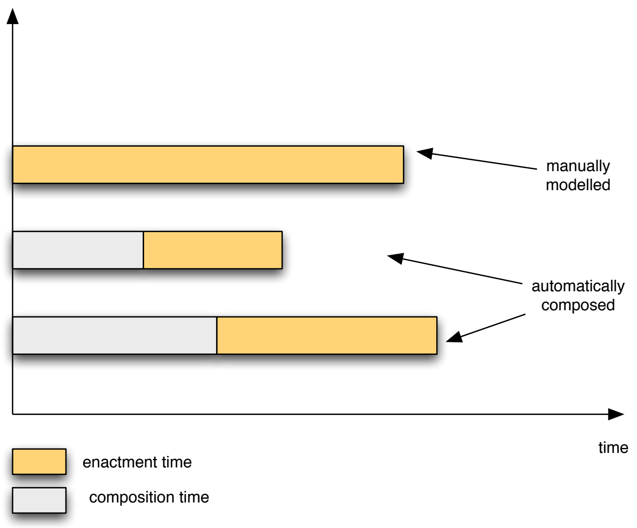


Figure 15: Possible advantage of automated service composition

Similar assessments must be performed in regard to other quality of service properties. While automated service composition reduces the costs for the enactment of individual compositions, it yields preceding costs for modelling ontologies and semantic service specifications.

To ensure this requirement two different approaches are used simoultaneously. Firstly, we must ensure that automated composition has as few drawbacks as possible. Composition time and modelling efforts should be as low as possible. Secondly, we must assess the usage of automated service composition individually for new scenarios. For this purpose the possible advantages (e.g.: flexibility, reduced enactment time / cost) and disadvantages (e.g.: modelling cost, composition time) must be reviewed in the context of these scenarios. It should only be applied were it gives advantages.

Req. 33: Flexible reaction to unexpected events (mandatory) The final requirement for the service composer is about possible reactions to unexpected events. Unexpected events are for example errors in service invocation, violation of non-functional properties. From a more general perspective unexpected events are not always bad. The addition of a new service that reduces enactment time is an example of a positive, yet unexpected, event[1].

In general failure handling in workflow management and service composition is done through exception handling [10, 1]. An exception handler defines activities to perform in case of specific failures. This approach has two drawbacks. Firstly, it is inflexible. Failure handling is only possible if an exception handler for this failure is provided. Secondly, all these failure handlers are modelled manually. This leads to complex and hard to understand service specifications.

To solve this problem of inflexible, manually modelled exception handlers one could compose these exception handlers automatically, too. But this leads to complex service compositions and works only if all possible exceptions are defined and described semantically. In ASG another approach is used. During composition exceptions are not taken into account. Instead a composition is created that only works if no exceptions happen (optimistic approach). If an exception happens,

[1]Of course ÃƒÂt is not unexpected that services are registered. But the registration of a specific service with distinct functionality cannot be expected.

service enactment terminates enactment. It then hands the current enactment status over to a component called *Mediated Replanning*. This component uses the enactment status to create a new service request for which the composer creates a new composition. The new service request has the same goal as the original request. The initial state of the request is adjusted according to already invoked services. It must be ensured that still running service invocations are reflected in the new service request and composition. Therefore the initial state of the new service request incorporates the effects of the running invocations and the new composition contains the running invocations[16, 6]. The composer must support this concept, called *re-composition*.

6 Conclusion

In this document we presented the requirements on the service composer in the ASG project. The requirements were derived and justified using scenarios developed by the work component C-7 *Usability and Demonstration*.

All together 33 distinct requirements have been identified. This includes four general requirements like the automation of composition, 25 functional requirements and 4 non-functional requirements. Functional requirements have been categorised according to a predefined schema.

The next steps after requirements analysis are the development of the service composition algorithm and language according to the defined requirements. Table 6 gives a summary of these requirements.

General Composition Requirements		
1	Service Composition is automated	mandatory
2	Service request describes the goal of composition	mandatory
3	Service Composition according to service request	mandatory
4	Information gathering services invocable during composition	optional
Functional Requirements		
Elements of Composition Requirements		
5	Elements of composition are services interactions	mandatory
6	Elements of composition are service compositions	mandatory
7	Services have input and output parameters	mandatory
8	Service functionality is described semantically	mandatory
9	Services can have more than one precondition or effect	mandatory
10	Services can have disjunctive preconditions	mandatory
11	Services can have disjunctive effects	mandatory
Control Flow Requirements		
12	Composition of sequential control flow	mandatory
13	Composition of parallel control flow	mandatory
14	Composition of alternative control flow	mandatory
15	Composition of multiple instances of sub-compositions	optional
16	Composition Language supports workflow patterns	mandatory
17	Composition is block-structured	optional
Data Flow Requirements		
18	Activities exchange data	mandatory
19	Activities use process input data	mandatory
20	Data exchange implies control flow	mandatory
21	Activities create new variables	mandatory
22	Scoping of process data	optional
Data Model Requirements		
23	Data elements are typed	mandatory
24	Data element types are defined in an ontology	mandatory
25	Composer is aware of data element structure	mandatory
26	Data elements can be used to evaluate control flow conditions	mandatory
27	External Application data exists	optional
Quality of Service Requirements		
28	Composition uses static non-functional properties	optional
29	Composition fulfils quality of service properties from service request	optional
Non-functional Requirements		
30	Service composition language bases on WS-BPEL	mandatory
31	Services are specified using the ASG Service Specification Language	
32	Service composition time must not exceed advantages from automation	mandatory
33	Flexible reaction to unexpected events	mandatory

Table 1: Summary of Requirements

Bibliography

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services – Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, 2004.

[2] Christer Bäckström. Computational aspects of reordering plans. *Journal Of Artificial Intelligence*, 9:99 – 137, 1998.

[3] Bill Curtis, Marc I. Keller, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75 – 90, 1992.

[4] Kutluhan Erol, James Handler, and Dana S. Nau. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, UMIACS-TR-94-31, ISR-TR-95-9, University Of Maryland, 1994.

[5] Kutluhan Erol, Dana S. Nau, and V.S. Subrahamnian. Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis. Technical Report CS-TR-2797, UMIACS-TR-91-154, SRC-TR-91-96, University Of Maryland, 1991.

[6] Michal Gajewski, Harald Meyer, Mariusz Momotko, Hilmar Schuschel, and Mathias Weske. Dynamic failure recovery of generated workflows. In *Proceedings of the 16th International Conference and Workshop on Database and Expert Systems Applications*. IEEE Computer Society Press, 2005.

[7] Malik Ghallab, Dana Lau, and Paolo Traverso. *Automated Planning - theory and practice*. Morgan Kaufmann, 2004.

[8] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the Association for Computing Machinery*, 42(4):741 – 843, 1995.

[9] Ugur Kuter, Evren Sirin, Dana Nau, Bijan Parsia, and James Hendler. Information gathering during planning for web service composition. In *Proceedings of the Third International Semantic Web Conference (ISWC)*, number 3298 in Lecture Notes of Computer Science. Springer, 2005.

[10] Frank Leymann and Dieter Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall, 2000.

[11] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *Journal on Artificial Intelligence Research*, 20:379 – 404, 2003.

[12] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL)*, 2004. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.

[13] Bernhard Peissl et al. ASG Based Scenarios in Telecommunications, Telematics and Enhanced Enterprise IT. Deliverable D7.I-1, NIWA Web Solutions, 2005.

[14] Dumitru Roman et al. Requirements Analysis on the ASG Service Specification Language. Deliverable D1.1-1, DERI Innsbruck, 2005.

[15] Earl Sacerdoti. The nonlinear structure of plans. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 206 – 214, 1975.

[16] Hilmar Schuschel and Mathias Weske. Triggering replanning in an integrated workflow planning and enactment system. In *Proceedings of 8th East-European Conference on Advances in Databases and Information Systems*, volume 3255 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 2004.

[17] Mithun Sheshagiri. Automatic composition and invocation of semantic web services. Master's thesis, University of Maryland, 2005.

[18] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5 – 51, 2003.

ISBN 3-937786-81-3
ISSN 1613-5652