Iterative Service Orchestration based on Dependability Attributes

Joachim Götze, Jochen Müller, Paul Müller
University of Kaiserslautern
PO Box 3049
D-67653 Kaiserslautern, Germany
{j.goetze, jmueller, pmueller}@informatik.uni-kl.de

Abstract

In a service-oriented architecture (SOA), the orchestration of services to new services and complex workflows is a common approach. Because the complexity and capability of the orchestrated services are increased, it is important to maintain and ensure the dependability attributes, e.g. availability and reliability, in such an environment. Today, most approaches focus on the needs during the creation of a single application, but the Internet became a commodity for end users during the last years and more services are available every day. While the range of available services is growing, the end users are helpless selecting a service providing the quality of service they expect. Therefore, a straightforward approach is needed allowing the creation and modification of orchestrated services by end users with respect to dependability attributes.

1 Motivation

Service-oriented architectures (SOA) – and Web services as a technology to implement a SOA – are emerging as a powerful concept to integrate current applications within and across organizational boundaries. More and more companies are using services in order to provide applications and business related data to their associates. Additionally, it became customary to enable not only business associates but also experienced end users to gain access to these or similar services. End users can now facilitate these services to create and provide their own orchestrated services. In the beginning the focus of these end user services is on providing an enhanced service, while later on the demands will be growing and the quality of the provided service will become more important.

As a result, the need to provide services and workflows of high quality increases the importance of dependability. The environment has to compensate for the continuously emerging and disappearing of services and service providers. Therefore, flexibility and transparency are key factors of successful service provisioning.

In an open environment, like the Internet, with a lot of potential service providers, there will be competition amongst the service providers. The hiding of service orchestration details is of huge importance. As a result, the information which services are used to create a new service and sometimes even what kinds of services are used is to be kept secret. Therefore, several levels of transparency need to be defined to allow for hidden confidential information, but also to reveal only as much information as needed to orchestrate a service.

The model proposed here describes an approach that allows the flexible orchestration of services. The orchestration process is simple and provides an easy way to create dependable services. The model focuses on dependability attributes enhancing the usage experience perceived by the user. Although other dependability attributes are important for a sound overall quality, these attributes are not included in the dependability analysis. During the orchestration process various levels of transparency are available, so that a service provider has the option to take preliminary actions to hide business secrets on the application level.

The prototype implementation of the concept proposed here is based on the Venice Service Grid [8] and is used within the project “DASIS” (Dependability and Adaptivity of Service-oriented Information Systems)\(^1\).

Section 2 outlines the work related to service dependability and compares these approaches to the model proposed here. Section 3 describes the concept proposed in detail and explains how orchestrated services are created and modified, which dependability attributes are taken into account and how the usage profile allows the improvement of a service during the modification process. In section 4 some resulting consequences and limitations of the concept

\(^1\)DASIS is part of the Cluster of Excellence "Dependable adaptive Systems and Mathematical Modeling" funded by the Program "Wissen schafft Zukunft" of the Ministry of Science, Education, Research and Culture of Rhineland-Palatinate, AZ.: 15212-52 309-2/40 (30)
are discussed. Section 5 outlines future work and concludes the paper.

2 Related Work

Reliability theory for hardware components is not transferable directly towards software components. Although software reliability is well-developed, the original focus of the research has been on large closed systems. Open distributed systems have to tackle new challenges by means of new solutions.

Many approaches focus on the determination of dependability before the system is shipped, e.g., the Cleanroom method [11], [5] advocating statistical testing. This approach is neither compositional nor does it allow for quick modifications of the system.

The need for usage profiles was mentioned in [11] and [7]. Neither of these solutions allow for quick system modifications and especially [7] requires access to the applications source code, which is not acceptable in an open distributed system based on the Internet.

[3] and [14] focus on the architectural structure, which is derived from low-level control compositions for functional components, e.g. sequences and pipelines. Again the information detail is not acceptable for an open distributed system based on the Internet and requires too much detailed knowledge to be suitable for end users. [13] follows a similar approach for component-based software architectures, but requires the modeling of usage profiles as Markov chains. The approach of [4] uses the assessment of UML-based software models. Again, requiring such detailed information is not acceptable.

Driven by the demand to automatically select high quality services for new workflows several approaches have been under consideration. In [15] a model for Web service composition is described using dependability attributes and reputation as selection criteria. The optimization approach focuses on finding an optimal solution by computing an optimal execution path in a statechart resulting in a high computation cost for the global optimization process for service selection. This approach requires the modeling of every composite service via a statechart that has to be available to allow for improvement of the service. In contrast to the approach proposed here the statechart of a service is only available to the original designer and therefore limits optimization of the service only to the designer.

In [2] the selection of services in a static linear workflow is analyzed. The approach includes replanning algorithms in order to improve the selection during the workflow execution, but again the ability to optimize the workflow is limited to the original developer.

[9] evaluates a completely different approach based on genetic algorithms for the service brokering based on quality of service attributes, but the authors state that their approach needs further investigation because of a reasonable difference of their results towards common branch-and-bound optimization approaches.

3 Model

The orchestration of services to new services and complex workflows is very common in service-oriented architectures. Resulting in increased complexity, but also in increased capability of the orchestrated service. Hence, the dependability of an orchestrated service or a workflow is affected by the dependability of the services integrated within. The availability of a huge number of services enables even end users to provide their own services created by themselves or by a mashup of already existing services. The demand of experienced end users is going more and more beyond a best effort and towards a dependable service provisioning.

The focus of the model proposed here is the orchestration of services for experienced end users. Typical approaches to software dependability focus on in-house applications or analysis of software prior to final shipment. While these software systems and the services available are in a more static environment, the service provisioning via the Internet has additional requirements, e.g., the regular integration of new services and the substitution of out-dated services based on a continuous fluctuation of available services.

Additionally, an externally provided service does not bring along an internal specification. If this would be an requirement, the amount of available services might be extremely limited as might be the number of service providers willing to share their intellectual property. This intellectual property could then be used to create a service like that by every other service provider. It might be extremely difficult to maintain a competitive advantage on this basis. Therefore, a solution is needed allowing the external service providers to keep and protect their intellectual property and to simultaneously create complex and transparent services. Such a solution will be explained in the following subsections. Then, the basic terms, which are a prerequisite for the model, will be identified and explained in detail. An example workflow is introduced in order to provide a thorough understanding of the concepts contained in the model proposed here.

Services are divided into two categories of services: composite services, i.e. the services that require additional (external) services to provide their functionality, and atomic services, i.e. the services that provide their functionality on their own (without additional services). A complex service consisting of one or many services provides the functionality for a single task within a workflow.

In order to demonstrate the basic terms and the concepts
of the model, the workflow of a travel agency (figure 1) is an ongoing example. The whole process includes the booking of a flight, a hotel and – depending on the distance between airport and hotel – a car or bike. Obviously the different workflow tasks, e.g. flight booking, are rather huge and include several sub-process, e.g. checking for available travel routes, that are hidden within this building block.

The following sections show the different service-usage profiles (ensuring a blackbox view onto the services), how composite services are orchestrated, and how the dependability attributes are combined to provide overall information about a composite service, and how some of the external services used by a composite service can dynamically be changed to improve dependability during runtime.

3.1 Service-Usage Profile

In order to create workflows using services that meet a well-defined quality of dependability, it is important to select services of adequate dependability to achieve the desired goal. This can only be done, if information about dependability of every service included is available. Therefore, in this model every atomic service has to provide information about its dependability attributes and every composite service has to provide additional information about the external services that are used to provide the desired functionality.

In the model proposed here, the internal structure of composite services is kept simple and not as complex as a workflow description. A service only exposes the service-usage profile: the required external services, and the type of these services, i.e. the functionality that these services provide, and the service-usage probability, i.e. how likely the use of each external service is, and the service operator (s. section 3.2). The required external services can be composite services themselves, hence the external services are building a tree structure of involved services. This tree structure is not a globally known information or stored in a centralized way, but can be created by following recursively the required external services from the top-level composite service.

The service-usage probability only quantifies the use of an external service during the average utilization of the composite service. Therefore, it does not depend on a selected function and there might be service operations that do not require a certain external service at all. The usage probability is absolutely vital in order to compensate for the missing information about the data flow within the composite service and by this allows the identification of the important and the more dispensable external services. The statistical information that is needed to provide the service-usage probability can be measured by every service in an autonomous way, e.g. by using a unique call function for external services accounting every service request.

This still might conflict with the interests of most service providers, because typically the internal algorithms of a service are kept secret in order to protect intellectual property. Hiding the internal structure of a service prevents competing service providers to easily identify weaknesses and strengths of a service in order to improve their own services. Nonetheless this transparency would also allow the service user to identify the weaknesses and strengths of a service, so that the service user is able to select a service of adequate quality complying with their requests.

Therefore the model proposed here allows three different nuances of transparency to satisfy the needs of service providers as well as the needs of service users:

- **No transparency** allows the service provider to completely hide the internal structure of a service. For the service user, a service providing no transparency has the same characteristics as an atomic service. Although the service user has no possibility to assess the internal structure, the service still provides its dependability attributes. This transparency level provides the highest protection for intellectual property, because not only the internal algorithms are kept secret, but also any external services that are utilized.

- **Partial transparency** means that the internal structure of a service is published by the service provider. The service user is able to identify all services involved in providing the functionality of the service and therefore the service user is able to use the dependability attributes of the involved services to predict the overall quality of the different attributes. This transparency level protects only the internal algorithms, but the use of external services might provide hints which kind of algorithms are used. Considering the gained confidence of the user, because the user does not only have to accept the presented dependability information, but has the option to observe them themselves, this can be an acceptable solution for service providers.

- **Total transparency** includes not only the revelation of the internal structure of a service, but also allows the user to choose any involved external service themselves to adjust the quality of dependability to their personal needs. These adjustments are only optional.
and the service provider can assign default services to ensure the functionality of the service and to ease the effort of use for an undemanding or inexperienced user.

Obviously every level of transparency includes the characteristics of the less transparent levels. Therefore, in the following sections total transparency is always assumed. This does not alter the quality of the results, because only the reproduction of the provided information is restricted in lower levels of transparency. A level of no transparency is comparable to the use of an atomic service and a level of partial transparency is comparable to total transparency except the ability that the user can change the involved services themselves. Especially in a closed environment, e.g. a company’s intranet, total transparency is expected to be the best choice to enable high-quality workflows with predictable and adjustable dependability for experienced users.

3.2 Service Orchestration

In this model, the description of the internal structure of a service (service-usage profile) includes several aspects. First, the dependability attributes of the service itself without influences from the involved external services. Second, the list of external service types. Each of these service types specifies the functionality of a service that is needed for the service’s operation. Third, a probability value (service-usage probability) for every service type providing an estimation of how often a certain service type is used when the service itself is requested. The fourth aspect is the service operator allowing not only to define a single genuine service for every service type, but to combine multiple services of the same type to enhance certain dependability attributes, e.g. using redundancy to increase availability. This combination is done by the three operators ONE, ANY, MAJORITY which are explained in the following:

- **ONE** allows to define multiple services of the same type, but only one of them is selected at random to provide the service. If this service should fail to reply, another service is selected at random. This operation allows to increase availability and to minimize cost at the expense of performance, because in the worst case many attempts might be needed to utilize a service. Additionally the random selection restricts this operation to services that are completely stateless.

- **ANY** allows to define multiple services of the same type, which are all used for the same request. This operation uses the result returned by the first replying service. Any result of the services replying afterwards are discarded. This operation allows to enhance availability and performance but ignores the cost for using these services.

- **MAJORITY** is an operation that schedules the same request to all defined services. Afterwards this operation uses the results of all services that did not fail and chooses the most common result. This operation enhances reliability and availability at the expense of cost and performance.
car rental service. Therefore the service operator ONE is used. As a result another service is used to access the customers financial information if the request sent to the first service has failed. This redundancy only handles failures in service requests, but does not take action if the first service does not find any financial information of a customer.

3.3 Reliability and Availability

Software reliability is described by the continuity the service is provided correctly, while availability is the readiness of the service to accept and process requests [1]. Reliability is a function of the software faults and its operational profile, i.e. the inputs to and the use of the software, and for open systems it is also a function of the reliability of essential required services [13].

The two terms reliability and availability are generally based on the terms mean time to failure (MTTF) and mean time to repair (MTTR). MTTF describes the average lapsing time until the next failure happens, while MTTR describes the average time needed to restore normal operation. The reciprocal of MTTF is the failure rate λ and the reciprocal of MTTR is the repair rate μ. Availability is a measure of the service accomplishment with respect to the alternation between the states ready for service and interrupted and therefore defined as

\[
Availability = \frac{MTTF}{MTTF + MTTR}
\]

MTTF and MTTR are typically measured for hardware systems. For software systems, the failure rate λ is more commonly used. While hardware systems need manual interaction in order repair a detected failure, the MTTR of a software system is very short, e.g. time of system restart or micro-reboot of a component. Since a service is not running all the time, the reliability is measured relative to execution time or to the number of calls. Therefore, availability ranges from 0% to 100%, with 100% being perfect and reliability is measured by the failure rate λ ∈ [0, ∞], i.e. failures per service request [10] [6] [12].

For a composite service utilizing multiple external services, the execution of a service request can be distinguished into multiple states: the request process of the service, and the local execution of functionality of the requested service, and the execution of functionality in several external services (figure 4). The reliability of a service request can be modelled with two types of components:

- The request process of the service containing the service call and the return of the results is dominated by the quality of the network and its connections. Because of the complexity to identify the reliability \( r_n \) of the underlying network of the analyzed service environment, i.e. the Internet, an educated guess must suffice for the quantification.
- The reliability of the local execution \( r_l \) is dependent on the length of the code, the type of operations performed, the underlying libraries, the operating system it runs on, and the hardware used. The quantification of \( r_l \) can be done for example by service monitoring.
- The reliability of the execution of external services \( r_{e_i} \) depends on the selected external services \( e_i \) used to provide the required functionality. The reliability of external services relies on separate code and other underlying systems. The quantification of \( r_{e_i} \) can be done by service monitoring and is provided by the external services themselves (s. 3.1).

A failure-free execution must run through all these states, therefore, reliability of a service request \( r_s \) can be modelled as a product of separate reliability factors:

\[
r_s = r_n r_l \prod_i r_{e_i}
\]

The availability and reliability \( r_s \) of the services \( S = \{s_1, ..., s_n\} \) combined by the operators ONE, ANY, MAJORITY has to be calculated at first. The resulting reliability of all three service operators is

\[
r_s = 1 - \prod_i (1 - r_s_i) = 1 - \prod_i \lambda_i
\]

A service operator only fails, if all the services combined by the operator fail. The aspects distinguishing these operators are the influenced side-effects of improved performance or cost. The service operator MAJORITY is special in comparison to the other operators, because it allows the service requester to combine for example services underlying measurement errors, i.e. the result is reliably provided, but the result itself is false, e.g. in sensor networks. Afterwards the overall availability and reliability of the composite service can be calculated by the above mentioned equations.

Figure 4. Different states of a service request

A prerequisite for the use of the equations to calculate a combined MTTF or a combined failure rate is the statistical independence of the services involved. In a large
environment with multiple service providers, this statistical independence can not be expected or even guaranteed. Typically, there are services that are commonly used by a large number of people or services, e.g. authentication and authorization services. If such a service might fail, this failure might influence multiple required external services. Also huge service providers are expected to offer multiple services running on the same host resulting in a dependence of all these services on their host machine. These services induce a statistical dependence into the system, which is also done by the communication basis: the network. Multiple hosts of a service provider might be located in the same network segment resulting in a dependence of all these services on their local network. These and other circumstances induce a statistical dependence into the system of services, which influence the quality of the calculations of availability and reliability.

The focus of this model is the orchestration of services and the influence of dependability onto composite services. Therefore the statistical dependence between several services that are orchestrated directly or indirectly into a new composite service need to be measured. A directly orchestrated service is a service that is used directly by the new composite service, whereas a service is indirectly orchestrated if it is providing its service to the new composite service through other services that are either directly or indirectly orchestrated.

In order to measure the influence of a certain service onto the composite service as a whole, the service-usage probability of all external orchestrated services used by the composite service are utilized. These probability values allow the calculation of the expected impact of the service in relation to the whole set of orchestrated services. Although the probability values do not allow to correct the calculated reliability and availability, they provide a hint about the extent of the influence. Based on this information the user can decide to change certain external services in order to either improve the dependability or to decrease the risk resulting from the influence of these services.

Based on the example, the overall availability and reliability of the service *rent a car* including all external services can be calculated using the values in table 1. The reliability of the complete service can be calculated using the combined reliability of the service *car preferences* and the service *car database*.

\[
\begin{align*}
    r_{S_1} &= (r_{N_{S_1}}) \prod_{i=2}^{4} r_{S_i} = \\
    &= (r_{N_{S_1}}(r_{N_{S_2}})(r_{N_{S_3}})(r_{N_{S_4}})) = \\
    &= \left(\frac{9999}{10000}\right)\left(\frac{9999}{10000}\right)\left(\frac{9999}{10000}\right)\left(\frac{9999}{10000}\right) = \\
    &\approx \frac{993}{1000}
\end{align*}
\]

The resulting overall failure rate for the service *rent a car* is approximately \(\lambda \approx \frac{1}{142}\) or one failure per 142 service requests.

### 3.4 Confidentiality and Integrity

The dependability attribute confidentiality ensures the absence of unauthorized disclosure of information. The characteristic of integrity is the absence of improper system states or system alterations.

Confidentiality and integrity of a service are influenced by several aspects. In the model proposed here, these aspects are classified into three categories: communication risks, processing risks, storage risks.

The confidentiality and integrity of messages exchanged via network can be enforced by various algorithms. This is surely the most important aspect, because during the network communication any information is most vulnerable to unauthorized access and falsification.

Beyond the dangers of network communication, there are local aspects within a service itself. At runtime a service might create temporary data containing confidential information that is unprotected. Another local aspect is long-term data storage and the protection of the stored data. Depending on the service and the user not all of these aspects are always important or even existent, e.g. a service might not need temporary data or require long-term data storage.

In order to orchestrate services to the user’s needs, it is important to take into account the usage scenario and the modality of the service. Every service needs to use network communication to provide its service, whereas not every service creates temporary files or provides long-term data storage. Therefore a service should not only provide information about the available security mechanisms, but also if it is reasonable to expect security mechanisms to minimize local risks.

All of these aspects can play an important role depending on the usage scenario a service might participate in. In order to allow an evaluation, collections of possible methods...
to enforce confidentiality or integrity concerning both local and communication risks have to be created and maintained. If new methods are developed, these methods need to be added to the appropriate collections. A service provider needs to select the appropriate methods used in his service in order to allow an evaluation.

Finally, in order to allow the user to select appropriate services for orchestration, a service is needed that allows to map the provided mechanisms onto quantitative values in order to make calculations. These quantitative values enable an unexperienced user to gain a good idea of the quality of the selected services. Therefore, every collection needs to be organized into an hierarchical ordering. In order to get quantitative values, the hierarchical ordering is mapped onto the interval \([0, 1]\). As a result, in this model there are six scales - three for each aspect of confidentiality and three for each aspect of integrity.

In order to evaluate an attribute of an integrated service, all the participating services are taken into account. If the attribute is related to an optional local aspect, e.g. long-term data storage, then only the attributes of services which really need to provide confidentiality or integrity are taken into consideration. Because the quality of confidentiality and integrity is only as good as the weakest component, the overall value of a confidentiality or integrity aspect is the least value provided by any (relevant) participating service.

### 3.5 Maintainability and Safety

Maintainability ensures that a (running) system is able to undergo modifications and patches, either offline or at runtime. The safety of a system prevents severe consequences of a failure affecting the user and the software environment in which the service is provided.

Although maintainability and safety are important aspects of dependability, these two dependability attributes are much more difficult to put into a quantitative value. In the model proposed here, the focus is on the dependability observed by the user. Therefore, maintainability in this model is identified as an attribute providing information for the user about the regular times a service is available or the times the service is down for maintenance.

Safety definitely contributes to the overall dependability of a system, but from the user’s point of view this attribute is of rather less importance. From a service provider’s point of view this attribute is much more important, e.g. increased safety can isolate failures within his environment without effecting other services hence increase availability and reliability. The user recognizes an improvement in safety only through an improvement of the other attributes, therefore this attribute is not included in the model so far.

### 4 Consequences and Limitations

The objective of this work is to enable end users not only to utilize Web services to fulfill their needs, but also to allow for service orchestration in a dependable way. The orchestration of services based on dependability attributes in this model is currently done through manual orchestration, where services are selected directly by the user. The calculated dependability attributes and probability values of the resulting composite service are then used to manually optimize the composite service towards the user’s expectations. The final objective is another possibility for service orchestration based on this model: automatically selecting external services of a composite service for the user towards certain goals the user has defined beforehand.

Both approaches have to compensate the disadvantage resulting from the tree structure of the external services. The optimization problem using a tree structure is very complex, because changing an already selected service not only changes the dependability attributes of this service. The newly selected service might also need different types of external services to provide its functionality. Therefore changing an already selected service might typically include the selection of multiple new services that are directly or indirectly required by the newly selected service. The previously selected external services might not be adequate anymore, because the newly selected service might use a different approach to provide the expected functionality and might therefore need different external services.

![Figure 5. Service optimization in the travel agency example](image-url)
information provided by three other different databases. Because of the total transparency of the service any of these utilized databases can be exchanged by another database. The tree structure of the external services used by the service rent a car has changed and might require a new selection of external services to meet the users dependability requirements.

In order to achieve a high quality composite service including many other services, manual orchestration might take a long time, because of the many different possibilities that can be taken into account. The use of an orchestrating service selecting the external services towards a user specified goal is beneficial in any case.

While work towards an automatic optimization approach is still at the beginning, the model proposed here makes a number of limiting assumptions: (a) the availability of the data required to calculate the dependability data according to the described rules, (b) assumptions concerning the mathematical model.

Some of the required data can be gained through monitoring of the respective services, but others have to be entered by service providers, e.g. the mapping of security algorithms onto discrete numbers. The quality of the data is highly dependent on the regular maintenance of the service data.

Additionally, the independence of service failures is assumed. This assumption is now widely accepted for well-designed highly dependable software [5], but still error propagation can invalidate this.

5 Summary and Future Work

This paper described a model currently under research. The focus of the model is the flexible orchestration of services to allow the creation of composite services even by end users. In order to improve the dependability of these composite services, the model describes how dependability attributes are used to help the user select suitable services during the orchestration process. Core issues of this approach are the use of service-usage profiles, i.e. to provide top-level information about service internals, and service operators, i.e. allowing to enhance the dependability of a composite service by using multiple external services of the same service type in a simple way. The methods explained are practical, because the results can be obtained reasonably efficiently and relevant statistical data can be gained analytically from monitoring of services.

In the future the implementation of this model will be completed and thoroughly tested. Subsequent to the implementation, the quality of the results of this approach will be measured by an evaluation. Additionally an idea to ease the difficulty of creating services is considered using optimization techniques to automatically select external services in a service composition meeting the user’s needs. The extension of attributes is currently under consideration so that future implementations might use e.g. reputation as additional selection parameters for the orchestration process.

References