# Quality-Oriented Design of Services

Michael Gebhart, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{gebhart | abeck} @kit.edu

***Abstract*—With the shift to a service-oriented architecture, goals concerning the IT of an organization, such as an increased flexibility and maintainability, are expected to be attained. For this purpose, the building blocks of the service-oriented architecture, the services, have to be designed that certain quality attributes, such as loose coupling or autonomy, are fulfilled. Existing design processes for services name these quality attributes and consider them as important. However, they do not explain their usage within a design process in order to create services that verifiably fulfill these quality attributes. This article shows an enhancement of existing design processes that on the one hand comprehensibly describes how to derive service designs from artifacts of the business analysis and on the other hand integrates quality attributes in order to enable a verifiably quality-oriented design of services. The approach is applied to design services for a system at the Karlsruhe Institute of Technology that guides students across the campus of the university.**

***Keywords-service design; design process; quality attribute; design decision; soaml***

## I. INTRODUCTION

Today, several companies structure their information technology (IT) service-oriented, where functionality is encapsulated and provided in form of services. The shift to a service-oriented architecture is mostly associated with the achievement of goals concerning the IT, such as an increased flexibility and maintainability [3, 4, 24].

To support the achievement of these goals, quality attributes could be identified, a service within a service-oriented architecture should fulfill. Wide-spread attributes are a unique categorization, loose coupling, autonomy and discoverability of a service. After the analysis of the business, the services are designed before they are implemented. Thus, during the so-called service design phase, the IT architect has to design the services in a way that the implementation results in services that fulfill these quality attributes. The service design phase consists of two sub phases: the identification and specification phase [5]. Within the identification phase, service candidates as preliminary services and their dependencies are identified [3, 5]. Service candidates consist of operation candidates that represent preliminary operations. They constitute the structural basis for the following specification phase. During this phase, the service designs for each service are modeled. They describe the service interfaces for accessing the provided functionality and the service components that perform the functionality.

Existing design processes in the context of service-oriented architectures, as introduced by Erl [3], Engels et al. [4], the Rational Unified Process [19] for Service-Oriented Modeling Architecture (RUP SOMA) [5, 6, 7], and the Service Oriented Architecture Framework (SOAF) [8], focus on the steps that are necessary to design services at a high level of abstraction. They even name an excerpt of quality attributes and consider them as important. However, they do not describe how the design of the services has to be performed in order to verifiably fulfill the quality attributes. Additionally, the design processes are mostly only described abstractly, so that a detailed description about how to derive service designs based on a standardized modeling language from artifacts of the business analysis is missing. Other work, as introduced by Erl [9, 22], Engels et al. [4], Reussner et al. [10], Josuttis [11], Maier et al. [12, 13], Perepletchikov et al. [14, 15], Hirzalla et al. [16], Choi et al. [17] and SoaML [18], focuses on quality attributes a service should fulfill. However, the authors of this work do not address how these quality attributes can be used within a design process in order to create services with these quality attributes.

This article introduces an enhancement for design processes as they are introduced in existing work in order to verifiably design services with certain quality attributes. For this purpose, the derivation of service designs from artifacts of the business analysis is described in detail. Additionally, an iterative analysis and revision phase is added subsequently to the identification and specification of services for ensuring the fulfillment of certain quality attributes. During the analysis phase, the quality attributes of the current service designs are evaluated by measuring quality indicators that represent the quality attribute and give hints about their current value. Afterwards, if the quality attributes do not correspond to the desired values, the revision phase is performed. This phase consists of two steps. First, the design flaws within the current service designs are identified as they give the IT architect hints about the model elements within a service design that should be revised. Afterwards, action alternatives are derived and presented to the IT architect. They represent design decisions the IT architect should consider in order to create revised and improved service designs.

To illustrate our approach, services of a service-oriented system that guides students across the campus of the

Karlsruhe Institute of Technology (KIT) are designed. This system has its origin in a service-oriented surveillance system developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [33, 34] we already designed services for [1, 2, 32]. The services are designed with respect to loose coupling, autonomy, unique categorization, and discoverability as desired quality attributes. The service designs are modeled using the Service-oriented architecture Modeling Language (SoaML) [18] as standardized UML profile [38] and metamodel for describing and formalizing service-oriented architectures. Though SoaML is a very new UML profile and metamodel and still under development, it is becoming increasingly accepted and employed.

The article is organized as follows: Section 2 presents the fundamentals in the context of design processes, quality attributes, and modeling service designs. In Section 3, the entire approach for a quality-oriented design of services is introduced and exemplarily applied for designing services of the service-oriented KITCampusGuide. Section 4 concludes the article and offers suggestions for future research.

## II. FUNDAMENTALS

A quality-oriented design of services consists of three essential parts that have to fit together: First, the design process as framework for the entire quality-oriented design has to be specified. The design process describes the necessary phases within the design process and specifies the derivation of elements created during the business analysis into elements of the service design phase and transformations within the design phase. Additionally, the design process describes when and how to consider quality attributes to guarantee the fulfillment of quality requirements. The availability of measureable quality attributes constitutes the second part of a quality-oriented design of services. The quality attributes, such as loose coupling and autonomy, have to be described in a way that the IT architect can verifiably measure them. The modeling of service designs represents the third and final part of a quality-oriented design. The created service designs have to be modeled, i.e. formalized, that it is possible to evaluate them with respect to quality attributes and derive implementation artifacts as starting point for the implementation phase. Existing work mostly focuses on one of these three aspects.

### A. Design Processes

In [3], Erl introduces the service-oriented analysis and design phases that describe the steps necessary to design services. According to Erl, first, service candidates, the included operation candidates, and dependencies between these service candidates are identified. Afterwards, for each service candidate an entire service design can be created that specifies the service in detail. Even though the identification and specification is described, the comprehensible transformation of artifacts that have been created during the business analysis phase into service candidates and afterwards into service designs is missing. Also quality attributes, such as loose coupling, are considered as important but explained textually only. Information how to evaluate a formalized service design regarding these quality attributes in order to create service designs with verifiable quality attributes is not provided. The service candidates and service designs are also described using an own informal notation. There is no formal language used.

In [4] Engels et al. describe a method to derive services from prior described business services. For each business service a service within the service-oriented architecture is created. But also in this case, some quality attributes are only mentioned as important and not explained in a way that they could be measured on a formalized service design. Additionally, the design process does not explain how to use the quality attributes to gain services with certain quality attributes. Engels et al. also do not use a formal language to model created services. The services are mostly described textual.

The Rational Unified Process for Service Oriented Modeling and Architecture (RUP SOMA) as introduced by IBM [5, 6, 7] provides a detailed description about how to derive preliminary service candidates from prior modeled business processes and how to transfer these candidates into final service designs. However, also in this case quality attributes are only mentioned and not further considered. For modeling service candidates and service designs the proprietary UML profile for software services is applied [25]. But in current work [26], there is also a usage of the standardized SoaML introduced.

In [8], the Service Oriented Architecture Framework (SOAF) is introduced. This framework describes steps that result in services with the prescribed quality attribute business it alignment. The process does not consider own preferences. Information about how to transfer artifacts created during the business analysis phase into artifacts of the service design phase is missing and for modeling service designs an own notation is used.

### B. Measureable Quality Attributes

Other work focuses on the description of quality attributes and their measurement. Erl presents in [9, 22] design principles and patterns for services. These principles and patterns are explained in detail, but the concrete measurement on formalized service designs is not explained. Also the integration into an entire design process is missing.

Similarly, Engels et al. [4], Reussner et al. [10], Josuttis [11], Maier et al. [12, 13], Perepletchikov et al. [14, 15], Hirzalla et al. [16], Choi et al. [17] and SoaML [18] introduce important and partially even measurable quality attributes. But also in this case, the description of them is addressed. How to use these quality attributes in order to create quality-oriented service designs is not further explained. In [2] we presented the evaluation of service designs based on SoaML. This work already helps IT architects to evaluate service designs according to the informal description of quality attributes as described in existing work. In [1] we introduced how this measurement can be used for supporting design decisions within a design process in order to create improved service designs.

### C. *Modeling Service Designs*

According to Erl [3, 9, 22, 23] and IBM [5, 6, 7], the design process consists of two phases, the identification and the specification of services. During the identification phase, service candidates as preliminary services are identified. In a next step, final service designs are specified. Thus, to support the design process with a formal modeling language, the modeling of service candidates and service designs is necessary. Erl does not use any formal language whereas IBM uses an own proprietary UML profile for software services [25]. In the meanwhile, SoaML [18] has emerged as a standardized UML profile for modeling services within a service-oriented architecture. However, the SoaML standard does not explain how to use this modeling language within a design process and how to evaluate service designs that have been created using this language. SoaML supports several elements of service-oriented architectures. In the following, we introduce the modeling elements that are of interest in this article for modeling service candidates and service designs.

*1) Modeling Abstract Capabilities:* In SoaML a Capability element exists that represents a collection of capabilities. These capabilities describe the functionality a service provides. The Capability element is a stereotyped UML class, whilst the capabilities inside are modeled using operations. Additionally, dependencies between these Capability elements can be specified. They are modeled by means of usage dependencies and represent that a group of capabilities requires other capabilities to be performed. The following figure shows three Capability elements and their dependencies.

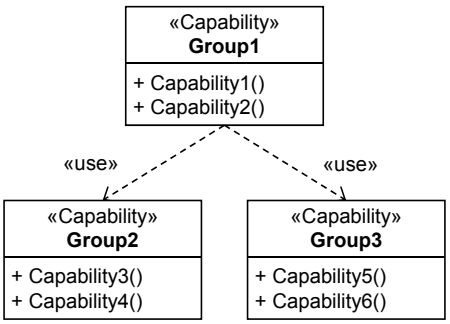


Figure 1. Modeling abstract capabilities

*2) Modeling Service Designs:* According to Erl [3, 23], Engels et al. [4], and IBM [6], a service design includes the design of a service interface and of a service component. The service interface describes the service and the service component realizes its functionality. To model a service interface, in SoaML the ServiceInterface element exists in form of a stereotyped UML class. A ServiceInterface describes the operations the service provides for potential service consumers. This is specified by a UML interface that is realized by the ServiceInterface. Additionally, it includes a specification of operations a service consumer has to provide for example in order to receive callbacks. For that purpose a second interface has to be created that is used by the ServiceInterface. A ServiceInterface also allows the description of participating roles in form of UML parts and of an interaction protocol. Latter can be specified by an UML Activity that is added as OwnedBehavior to the ServiceInterface. An exemplary service interface is shown in the following figure. This service interface describes that one operation is provided and one operation is required to be provided by the service consumer in order to receive callbacks. The interaction protocol specifies the order of operation calls for gaining a valid result.

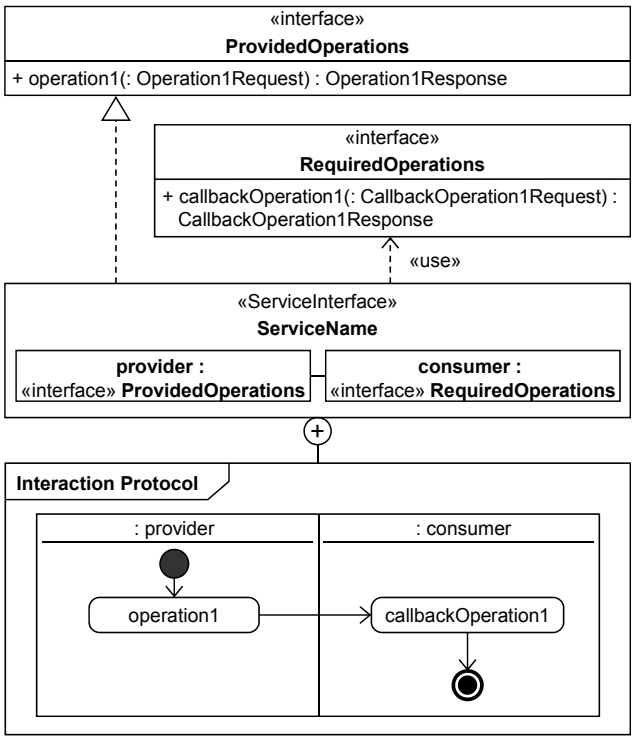


Figure 2. Modeling a service interface

When calling one of the provided or required operations, messages are exchanged. These messages are described by MessageType elements that extend the UML dateTypes. A MessageType represents a document-centric message and can contain several dataTypes. In context of specifying service designs, also these messages have to be described. In the following, an example for a message is depicted.

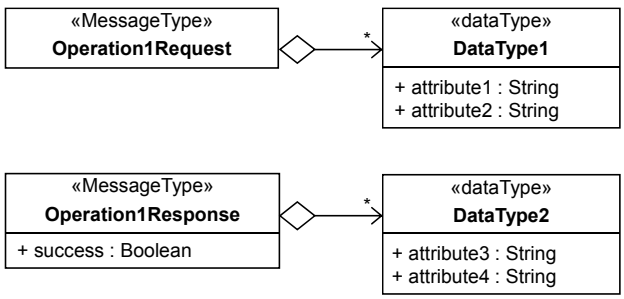


Figure 3. Modeling message types

Finally, for each service design the service component has to be specified that realizes the provided functionality. For service components, SoaML includes the Participant element that represents an organization, system, or software component. For modeling a Participant in UML, the UML component can be extended by an according stereotype. For each provided service a ServicePoint is added and typed by the describing ServiceInterface. If the service component requires other services to fulfill its functionality, RequestPoints can be added to the service component. They specify required services and are also typed by the describing ServiceInterface element. To model the internal behavior of the service component, for each provided operation an OwnedBehavior in form of an UML Activity can be added. For each ServicePoint and RequestPoint a UML Partition is added that is typed by this ServicePoint or RequestPoint and for each operation of a service a CallOperationAction is assigned to the according Partition. An AcceptEvent describes that the service component waits for a callback operation being invoked. For internal functionality that is not performed by required services an OpaqueAction is added to the Partition that represents the ServicePoint. An exemplary service component is depicted in Figure 3. The service component provides one service and requires two services.

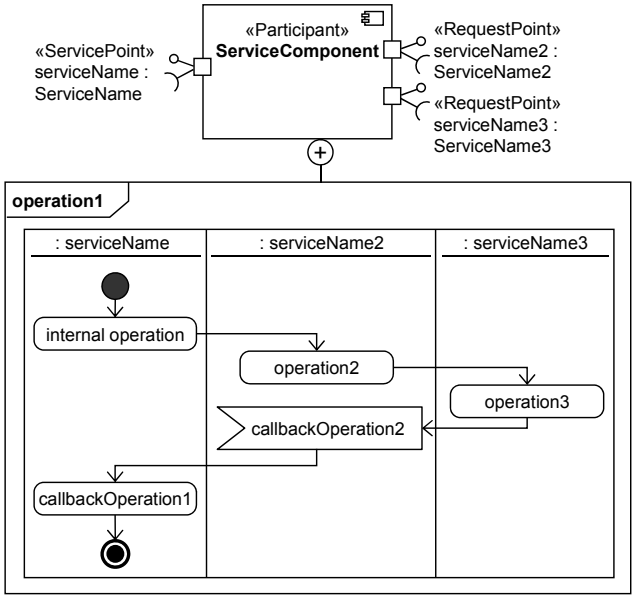


Figure 4. Modeling a service component

### D. Discussion

The analysis of the existing work shows that each work focuses mainly on one aspect. Work focusing on design processes describes the necessary steps within the process. The concrete derivation of service candidates and final service designs with a concrete modeling language is not addressed. Also quality attributes are only considered as important but it is not obvious how to measure them and how to use this knowledge to create quality-oriented service designs.

Other work focuses on exactly these quality attributes and shows metrics that enable their measurement. But in this case, it is not obvious how to measure the quality attributes on a standardized modeling language, such as SoaML. The textual descriptions have to be interpreted and the formalized metrics require information that is mostly not part of service designs. Finally, the quality attributes are not integrated into an entire design process. Thus, there exist only detailed descriptions of quality attributes but their usage to create service designs with certain quality attributes is missing.

Modeling languages for service designs, such as the UML profile for software services and SoaML, focus on modeling elements and do not provide any information about how to use this language within an entire design process. Only IBM gives some hints about how to derive artifacts from prior modeled business processes [6, 25, 26]. But how to use this language to model service designs with certain quality attributes and how to evaluate a modeled service design regarding these attributes is not explained.

Our quality-oriented service design approach combines these different approaches. We use the design processes as described in existing work and add additional phases for ensuring certain quality attributes. During these phases, our approach to evaluate service candidates and services designs based on SoaML [2] is applied. Afterwards, the service candidates and service designs are revised in order to improve chosen quality attributes [1]. Additionally, we add detailed information about how to derive service candidates in SoaML from modeled artifacts of the business analysis phase and how to transfer service candidates into service designs also based on SoaML. As result, a guideline is provided that enables the IT architect to comprehensibly create service designs with certain quality attributes.

## III. Quality-Oriented Design of Services

The design process of this article enhances design processes discussed in Section 2 by details about how to derive service candidates from artifacts of the business analysis phase and service designs from service candidates. Furthermore, subsequent phases for ensuring the fulfillment of quality attributes are added.

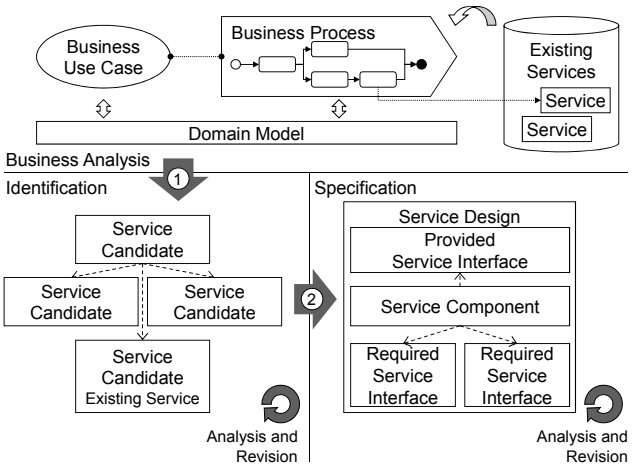


Figure 5. Design process

The design process requires a prior analysis of the business. This means that a domain model, business use cases and business processes are created. These artifacts are then transferred into preliminary service candidates as part of the identification phase. Afterwards, these service candidates are analyzed in regard to quality attributes. If the current attributes do not correspond to the preferred values, a subsequent revision is performed. During the specification phase, first, the service candidates are transferred into preliminary service designs. Also in this phase, afterwards, the service designs are analyzed in regard to quality attributes and if required revised. As result, service designs are created that fulfill certain quality attributes. Since the created artifacts of the business analysis phase constitute the basis for the design process, they are explained in the following.

### *A. Scenario*

To illustrate the artifacts of the business analysis and the subsequent design process, in this article the human-centered environmental observation domain referring to the network-enabled surveillance and tracking system as introduced by the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [33, 34] is treated. In this context the KITCampusGuide, a project at the Karlsruhe Institute of Technology (KIT) to provide a guide for students, lecturers and guest, is chosen as scenario. A person can ask for a person or a room on the campus of the university and the KITCampusGuide calculates the route. The following figure illustrates the scenario in action.

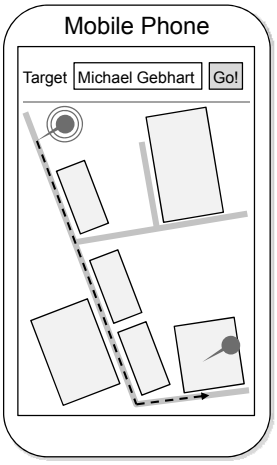


Figure 6. KITCampusGuide in action

The goal is, to create service designs for this scenario that fulfill the quality attributes of an unique categorization, loose coupling, autonomy and discoverability as introduced in [2].

### *B. Business Analysis*

During the business analysis phase, the following threw artifacts are created: The domain model captures all relevant concepts of the domain and their relations. It determines the relevant terms when designing the business processes and also unifies the terminology of the services. The business use cases describe the external visible business services that are expected to be supported by IT. The business processes describe the processes behind the business use cases, thus describes their implementation.

For modeling the domain, an ontology can be used. In this case, the ontology is created using the Web Ontology Language (OWL) [30] by means of Protégé [37]. As illustration, we choose a notation similar to the OntoGraf in Protégé. For each concept a rectangle is depicted and the relations between these concepts are represented by lines between them. If a concept or relation is available in various languages, this information can be added as labels. Each label can have a suffix specifying the language of the label, such as "@de" for German. An excerpt of the domain model for the human-centered environmental observation is depicted in Figure 7.

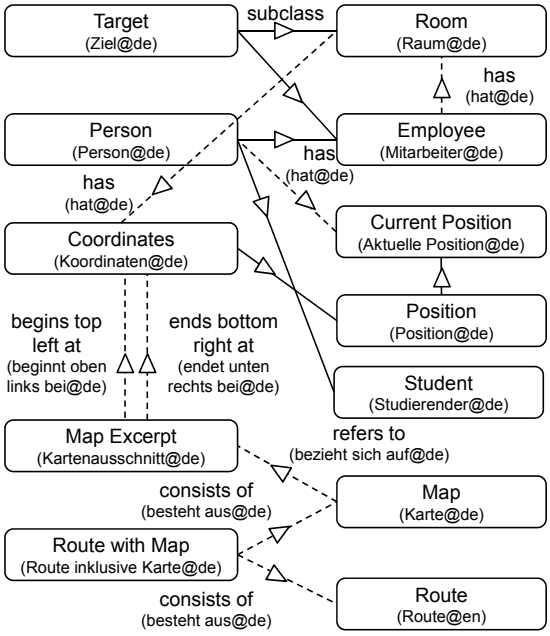


Figure 7. Excerpt of the domain model

The business use cases can be seen as entry points for the service design phase. They describe the externally visible business services [4] that are supposed to be supported by IT [6]. As notation, the UML profile for business services can be applied [20, 27, 28]. The use case describes that a student requests a route from his current position to a room or an employee. Additionally to the route, the map that covers the route is returned.

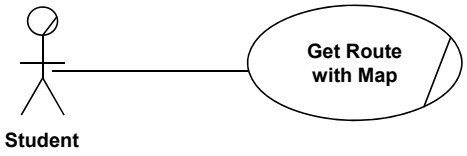


Figure 8. Considered business use case

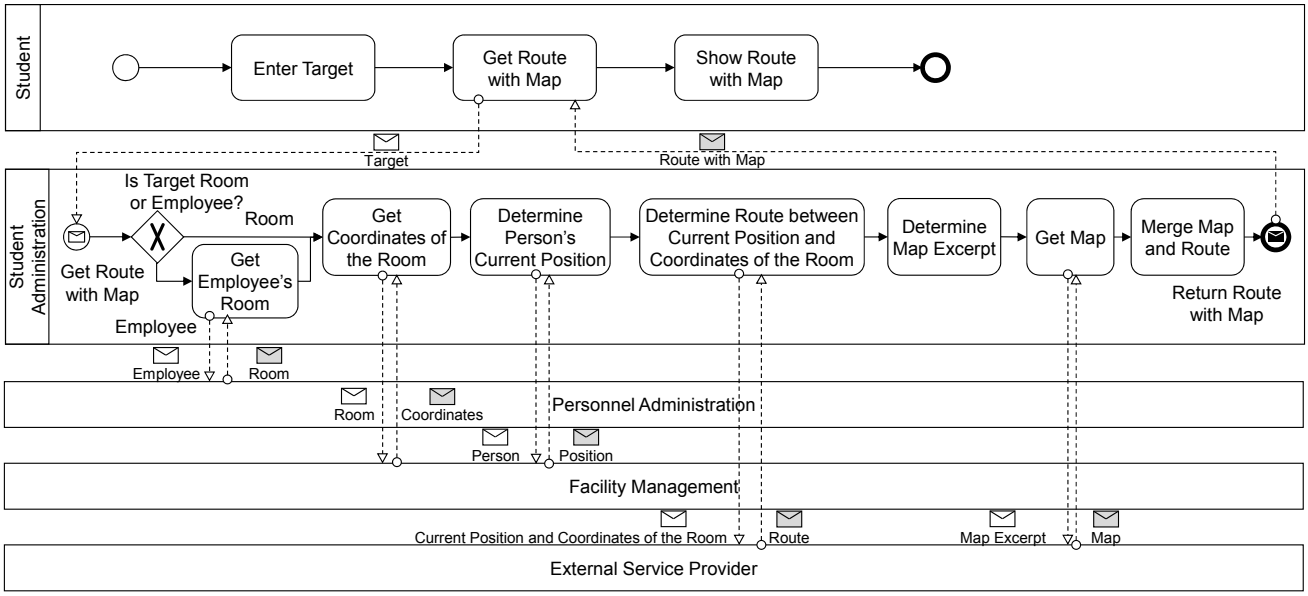


Figure 9. Business process to get a route with a map

Since each business use case or business service is realized by a business process [4, 6], the underlying business process has to be modeled. For this purpose the business process model and notation (BPMN) [29] can be used. The business process that realizes the considered business use case is depicted in Figure 9.

### C. *Service Design*

The service design phase starts with the identification of service candidates. According to Figure 5, the identification includes two steps: First, preliminary service candidates are derived from artifacts of the business analysis phase. Afterwards, these service candidates are analyzed regarding quality attributes and if necessary they are revised in order to improve the quality attributes. To derive the service candidates, the business processes are considered. For our scenario the business process as shown in Figure 9 is used to derive service candidates.

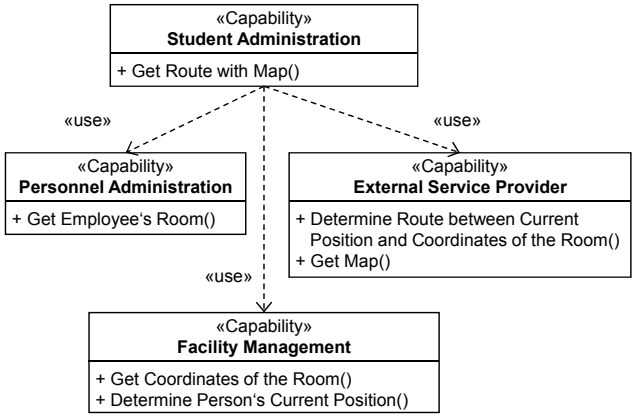


Figure 10. Derived service candidates

For each pool representing an organizational unit a new service candidate is created and for each message flow between the pools, an operation candidate is added. Since a service candidate represents a group of abstract capabilities, the Capability element of SoaML corresponds with the understanding of service candidates and thus can be used for modeling service candidates and their dependencies. Figure 10 shows the derived service candidates for our scenario.

To evaluate these service candidates regarding a unique categorization, loose coupling, autonomy and discoverability, the evaluable quality indicators as introduced in [2] are used and extended. For service candidates only a subset of the quality indicators is evaluable. The following table shows the quality indicators for each service candidate. A "+" represents that the quality indicator is optimal and a "-"describes that there is need for improvement. If a quality indicator is not evaluable, a "0" is set.

TABLE I. EVALUATION OF SERVICE CANDIDATES

| Quality Indicator | SA | PA | FM | ESP |
|---|---|---|---|---|
| ***Unique Categorization*** | | | | |
| Division of Business-Related and Technical Functionality | + | + | + | + |
| Division of Agnostic and non-Agnostic Functionality | + | + | - | + |
| Data Superiority | 0 | + | + | + |
| Usage of Common Business Entities | + | + | - | - |
| ***Loose Coupling*** | | | | |
| Compensation | 0 | 0 | 0 | 0 |
| ***Autonomy*** | | | | |
| Dependencies | - | + | + | + |
| Overlapping Functionality | + | + | + | + |

Since the service candidates only provide business-related functionality, the quality indicator to divide business-related and technical functionality is optimal for all service candidates. The division of agnostic and non-agnostic functionality can be improved for the Facility Management service candidate. Whilst the functionality to get coordinates of a room is very agnostic functionality, the determination of person's current position is very process specific and will not be used in many further scenarios. Since all service candidates are explicitly responsible for the management of used business entities, they fulfill the requirement for data superiority. Since the Student Administration does not manage any business entity, the data superiority is not evaluable for this service candidate. The operations of the Facility Management and of the External Service Provider do not use common business entities. The former uses the business entities room and person and both business entities can exist for their own. In case of the External Service Provider also different and independent business entities are used by the operations. Since there are no state-changing operations performed by any service candidate, there is no compensating functionality required. The Student Administration depends on other service candidates, thus the dependency quality indicator is not optimal. Since every service candidate is explicitly responsible for a functional scope, there is no functional overlap.

In a next step, the IT architect has to revise the service candidates, in order to improve their quality attributes. For that purpose, in a first step, design flaws in form of weak points have to be identified. They represent parts of the service candidate model that are responsible for a specific non-fulfilled quality attribute. Since each quality indicator refers to one main artifact within service candidates, this information can be used to identify the weak points. The following table lists the quality indicators and the model elements that represent the responsible part and thus the weak point.

TABLE II. WEAK POINTS IN SERVICE CANDIDATES

| Quality Indicator | Weak Point |
|---|---|
| Division of Business-Related and Technical Functionality | If at least the half of the operation candidates provide business-related functionality, then the operation candidates that provide technical functionality represent the weak point, else the operation candidates that provide business-related functionality. |
| Division of Agnostic and non-Agnostic Functionality | If at least the half of the operation candidates provide agnostic-related functionality, then the operation candidates that provide non-agnostic functionality represent the weak point, else the operation candidates that provide agnostic functionality. |
| Data Superiority | The operation candidates of other service candidates that manage business entities that are also managed by own operations represent the weak point. |
| Usage of Common Business Entities | First, the biggest set of used and depending business entities is determined. The operation candidates that use business entitites that are not part of this set represent the weak point. |
| Compensation | The operation candidates that provide state-changing functionality and do not have a compensating operation candidate represent the weak point. |
| Dependencies | The operation candidates that require other service candidates represent the weak point. |
| Overlapping Functionality | The operation candidates with overlapping functionality to operation candidates of other service candidates represent the weak point. |

The table above helps the IT architect to analyze the derived service candidates and to identify weak points that should be revised. Thus, for the Facility Management service candidate the operation candidate for determining person's current position represents a weak point, thus a design flaw, as shown in Figure 11.

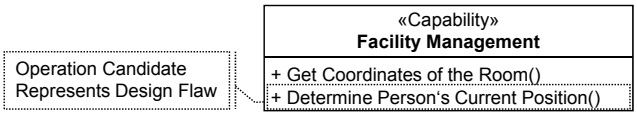


Figure 11. Identified design flaw

In a next step, the IT architect has to decide, how to revise this service candidate in order to fix the weak point. To support his decision, possible design decisions are analyzed and associated with the prior identified weak point. The quality indicators base on elements of service candidates or service designs that can represent weak points. Design decisions on the other hand influence these elements. This enables the association of design decisions with quality indicators. This association can be used to identify design decisions that affect certain quality indicators and can such be considered in order to improve weak points. The following figure shows the approach.

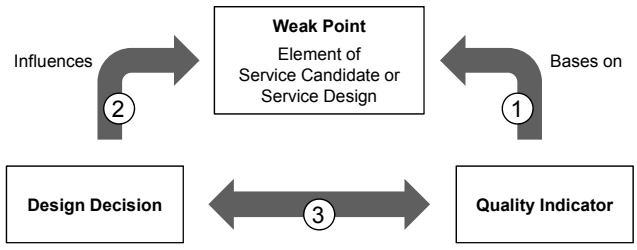


Figure 12. Approach for design decision identification

Possible design decisions can be taken from existing work that describes how to design services. Afterwards, these design decisions have to be adapted for a revision of service designs. For example, Erl describes in [3] that it is necessary to decide the operation candidates within a service candidate. Thus, a revision design decision is whether to move an operation candidate into another service candidate or not. For service candidates there are no further design decisions. During the specification phase there will be some more. The design decision whether to move an operation candidate can be further refined. The decision tree for our scenario including the various concrete action alternatives (AA) is shown below.

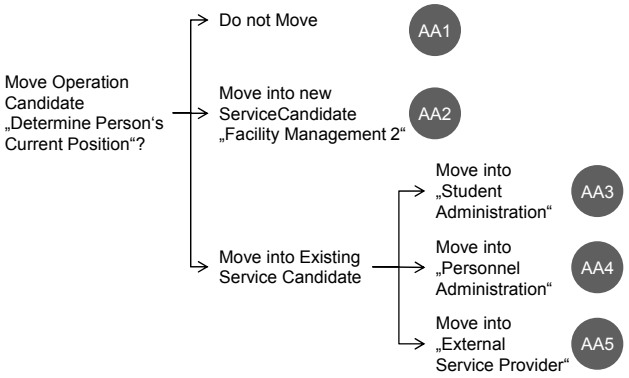


Figure 13. Action alternatives

It is important to notice that only possible concrete action alternatives can be considered that result in valid service designs. This means that for example the deletion of an operation candidate is not considered for this decision results in service designs that do not fulfill the business requirements. This restriction enables the convincing evaluation of different action alternatives.

Afterwards, each of the different action alternatives can be evaluated with regard to the quality attributes. This means that the service candidates are evaluated for each action alternatives. The following table shows this evaluation for each service candidate and the action alternatives two to five. For each quality indicator and action alternative, it is displayed whether it improves (↗), gets worse (↘) or does not change (➔) compared to the action alternative one that has been evaluated in Table 1. For the optional service candidate Facility Management 2, the new value is shown for there is no existing value that could be compared.

TABLE III. EVALUATION OF ACTION ALTERNATIVES

| Quality Indicator | | SA | PA | FM | $FM_2$ | ESP |
|---|---|---|---|---|---|---|
| Division of Business-Related and Technical Functionality | AA2 | ➔ | ➔ | ➔ | + | ➔ |
| | AA3 | ➔ | ➔ | ➔ | n.a. | ➔ |
| | AA4 | ➔ | ➔ | ➔ | n.a. | ➔ |
| | AA5 | ➔ | ➔ | ➔ | n.a. | ➔ |
| Division of Agnostic and non-Agnostic Functionality | AA2 | ➔ | ➔ | ↗ | + | ➔ |
| | AA3 | ➔ | ➔ | ↗ | n.a. | ➔ |
| | AA4 | ➔ | ↘ | ↗ | n.a. | ➔ |
| | AA5 | ➔ | ➔ | ↗ | n.a. | ➔ |
| Data Superiority | AA2 | 0 | ➔ | ➔ | 0 | ➔ |
| | AA3 | 0 | ➔ | ➔ | n.a. | ♏ |
| | AA4 | 0 | ➔ | ➔ | n.a. | ➔ |
| | AA5 | 0 | ➔ | ➔ | n.a. | ➔ |
| Usage of Common Business Entities | AA2 | ➔ | ➔ | ↗ | + | ➔ |
| | AA3 | ↘ | ➔ | ↗ | n.a. | ➔ |
| | AA4 | ➔ | ➔ | ↗ | n.a. | ➔ |
| | AA5 | ➔ | ➔ | ↗ | n.a. | ➔ |
| Compensation | AA2 | 0 | 0 | 0 | 0 | 0 |
| | AA3 | 0 | 0 | 0 | n.a. | 0 |
| | AA4 | 0 | 0 | 0 | n.a. | 0 |
| | AA5 | 0 | 0 | 0 | n.a. | 0 |
| Dependencies | AA2 | ↘ | ➔ | ➔ | + | ➔ |
| | AA3 | ➔ | ➔ | ➔ | n.a. | ➔ |
| | AA4 | ➔ | ➔ | ➔ | n.a. | ➔ |
| | AA5 | ➔ | ➔ | ➔ | n.a. | ➔ |
| Overlapping Functionality | AA2 | 0 | 0 | 0 | 0 | 0 |
| | AA3 | 0 | 0 | 0 | n.a. | 0 |
| | AA4 | 0 | 0 | 0 | n.a. | 0 |
| | AA5 | 0 | 0 | 0 | n.a. | 0 |

According to this table, action alternative two and four are the most improving ones. Now, the IT architect has to decide how to weight the quality indicators. In our case we decide that dependencies are less harmful. Thus, the IT architect chooses action alternative two. For the other weak points this procedure is repeated adequately. As result, service candidates are created that fulfill the four quality attributes best. The service candidates are displayed in the following figure.

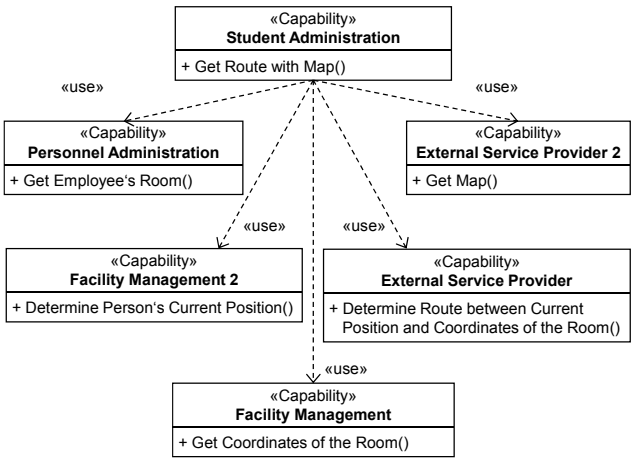


Figure 14. Revised service candidates

Subsequently to the identification phase, the specification follows. During this phase, first, preliminary service designs are derived and afterwards, they are revised if necessary. To derive the service designs, each service candidate is transferred into one ServiceInterface with one realized interface containing the provided operations and one interface containing required operations for receiving callbacks. The operation candidates are directly added as operations within the realized interface and if there is an end event within the corresponding business process that calls an operation, this operation call is added within the interface containing the required operations. Figure 15 shows the derived service interface for the service candidate Student Administration.

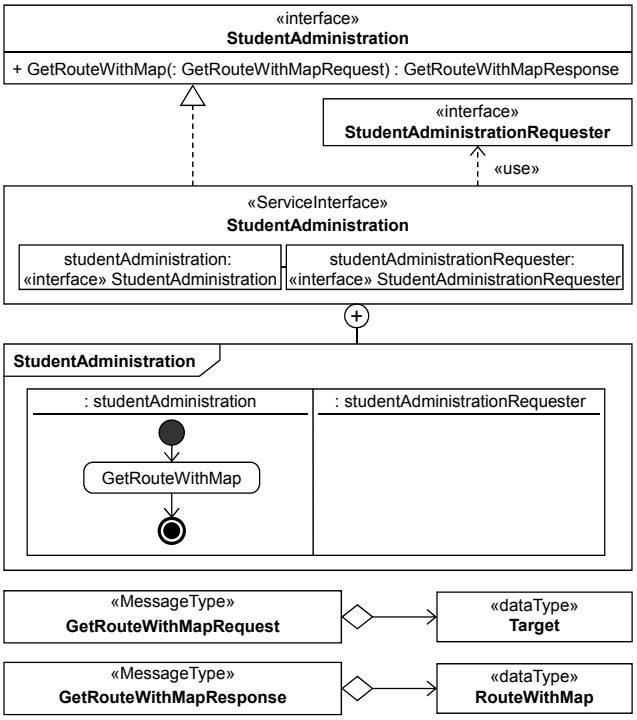


Figure 15. Derived service interface

For each operation adequate message types are created. They are named according to the operation with the suffix Request or Response. For the concepts exchanged as messages within the business process data types are generated and assigned to the respective message type. The interaction protocol of the service interface can be derived by the exchanged messages of the business process. All these artifacts are kept within a package named after the service interface. During this step already some information, such as potential naming conventions, can be considered. For example white spaces in the names of the service interface and the operations can be removed if this is a convention. Another convention could be the translation into another language. For example, if the business has been analyzed in German and it is convention to use English for service design artifacts, the artifacts can be translated according to the domain model and its labels containing the names of the concepts in various languages.

Additionally to the service interface, a service component is generated. The service component contains one ServicePoint typed by the derived ServiceInterface. If the service candidate where the service component was derived from requires other service candidates, appropriate RequestPoints are added. The following figure shows the service component for the Student Administration service candidate. The service component provides one service and thus includes one ServicePoint. To realize its functionality, ir requires five other services that are added as RequestPoints. Both the ServicePoint and the RequestPoints are named and typed by the ServiceInterface that describes the service. The internal behavior of the service component equals the business process, thus it is not further depicted.

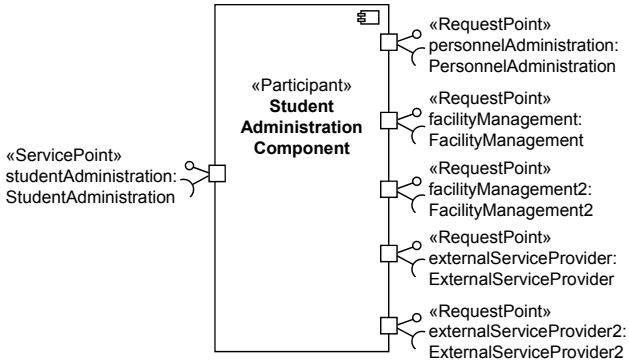


Figure 16. Derived service component

This systematic derivation is performed for all service candidates. Afterwards, the analysis and revision phase follows, similar to analysis and revision within the identification phase. During the specification phase, further quality indicators can be considered that were not of interest during the identification phase. The quality indicators are again taken from our previous work [2]. The service design for the Student Administration is evaluated in the following table.

TABLE IV. EVALUATION OF SERVICE DESIGNS

| Quality Indicator | SA | PA | FM | $FM_2$ | ESP | $ESP_2$ |
|---|---|---|---|---|---|---|
| ***Unique Categorization*** | | | | | | |
| Division of Business-Related and Technical Functionality | + | + | + | + | + | + |
| Division of Agnostic and non-Agnostic Functionality | + | + | + | + | + | + |
| Data Superiority | 0 | + | + | 0 | + | + |
| Usage of Common Business Entities | + | + | + | + | + | + |
| ***Discoverability*** | | | | | | |
| Functional Naming of the Service Interface | + | + | + | + | + | + |
| Functional Naming of the Roles | + | + | + | + | + | + |
| Functional Naming of the Operations | + | + | + | + | + | + |
| Functional Naming of the Parameters | + | + | + | + | + | + |
| Functional Naming of the Data Types | + | + | + | + | + | + |
| Naming Convention Compliance regarding the Service Interface | - | - | - | - | - | - |
| Naming Convention Compliance regarding the Roles | + | + | + | + | + | + |
| Naming Convention Compliance regarding the Operations | - | - | - | - | - | - |

| | | | | | | |
|---|---|---|---|---|---|---|
| Naming Convention Compliance regarding the Parameters | + | + | + | + | + | + |
| Naming Convention Compliance regarding the Data Types | + | + | + | + | + | + |
| Information Extent | + | + | + | + | + | + |
| ***Loose Coupling*** | | | | | | |
| Compensation | 0 | 0 | 0 | 0 | 0 | 0 |
| Asynchronity | + | + | + | + | + | + |
| Complexity of Common Data Types | + | + | + | + | + | + |
| Operation Abstraction | + | + | + | + | + | + |
| Data Type Abstraction | + | + | + | + | + | + |
| ***Autonomy*** | | | | | | |
| Dependencies | - | + | + | + | + | + |
| Overlapping Functionality | + | + | + | + | + | + |

According to this table, most of the quality indicators are already optimal due to the fact that the service designs were derived from already revised service candidates. Since the artifacts were generated from the service candidates that came from the business processes, also the naming of the artifacts already follows functional terms. Also most of the naming conventions have already been considered during the transfer of service candidates into service designs. Only the operations still do not follow our naming conventions. They should begin with a lowercased letter. This information could be considered during the derivation of the service designs too. But we consciously disregarded this convention in order to illustrate that the naming of the artifacts is an important aspect for the discoverability and even though some naming conventions have been already considered during the derivation of the service designs it should be reviewed and evaluated afterwards. Another example of naming conventions that cannot be fully regarded in an automatic transformation is the language of the artifacts. If the business has been analyzed in another language, such as German, the service candidates and the derived service designs are also in German. If the naming convention for the design artifacts is English, a translation is necessary. The domain model can contain several languages, so some information can already be used for an automatic transformation. However, mostly there is manual effort required for possibly not all concepts within the domain model are described in several languages. The quality indicators help to remind the IT architect that naming conventions, such as the correct language, have to be considered. The naming of the service interfaces is also not optimal. The reason is that a service interface should be named after what it is doing and numerations, such External Service Provider 2, should be avoided. A common naming convention for services that manage a certain business entity is to name this service interface after the managed business entity. For example, if the service manages the business entity room, a room service should be provided. Since all service designs contain all necessary information, the information extent is optimal. The asynchronity is optimal too, for there are no long-running operations that should be provided asynchronous. The complexity of common data types requires that all common data types are simple data types only. Since for each service design an own package has been generated, the complex data types are in separated packages and the service designs do not share any complex data types. On the one hand this requires a transformation of data types even if they are named equal, but on the other hand this supports the loose coupling. Since the operations hide the implementation and do not show any implementation details and the data types are only business-driven and not technical, the abstraction is also optimally fulfilled. The table shows that due to the systemic derivation of service designs from already revised and business-driven service candidates a lot of quality indicators are already optimally fulfilled. But the sum of quality indicators helps the IT architect to ensure that he has not forgotten any important aspect.

To revise the service designs, again the design flaws have to be identified and afterwards action alternatives have to be presented. The following table shows the weak points for the quality indicators considered during the specification of the service designs. At this, also the weak points that have been used during the identification phase are presented again, however they are adapted for the modeling elements within service designs instead of service candidates.

TABLE V. WEAK POINTS IN SERVICE DESIGNS

| Quality Indicator | Weak Point |
|---|---|
| Division of Business-Related and Technical Functionality | If at least the half of the operations provide business-related functionality, then the operations within the realized interface of the service interface that provide technical functionality represent the weak point, else the operations that provide business-related functionality. |
| Division of Agnostic and non-Agnostic Functionality | If at least the half of the operations provide agnostic-related functionality, then the operations within the realized interface of the service interface that provide non-agnostic functionality represent the weak point, else the operations that provide agnostic functionality. |
| Data Superiority | The operations within the realized interface of other service interfaces that manage business entities that are also managed by own operations represent the weak point. |
| Usage of Common Business Entities | First, the biggest set of used and depending business entities is determined. The operations within the realized interface of the service interface that use business entitites that are not part of this set represent the weak point. |
| Functional Naming of the Service Interface | The name attribute of the service interface represents the weak point. |
| Functional Naming of the Roles | The name attribute of the not functionally named roles represents the weak point. |
| Functional Naming of the Operations | The name attribute of the not functionally named operations represents the weak point. |

| | |
|---|---|
| Functional Naming of the Parameters | The name attribute of the not functionally named parameters represents the weak point. |
| Functional Naming of the Data Types | The name attribute of the not functionally named data types represents the weak point. |
| Naming Convention Compliance Regarding the Service Interface | The name attribute of the service interface represents the weak point. |
| Naming Convention Compliance Regarding the Roles | The name attribute of the not functionally named roles represents the weak point. |
| Naming Convention Compliance Regarding the Operations | The name attribute of the not functionally named operations represents the weak point. |
| Naming Convention Compliance Regarding the Parameters | The name attribute of the not functionally named parameters represents the weak point. |
| Naming Convention Compliance Regarding the Data Types | The name attribute of the not functionally named data types represents the weak point. |
| Information Extent | The service interface represents the weak point. |
| Compensation | Operations within the realized interface of the service interface that provide state-changing functionality and do not have a compensating operation represents the weak point. |
| Asynchronity | The communication modes of the CallOperationActions within the interaction protocol that correspond to operations with long-running functionality and are not asynchronous yet represent the weak point. |
| Complexity of Common Data Types | The data types that are complex and commonly used represent the weak point. |
| Operation Abstraction | The operations that are not abstract represent the weak point. |
| Data Type Abstraction | The data types that are not abstract represent the weak point. |
| Dependencies | The Operations within the realized interface of the service interface that require operations of other services represent the weak point. |
| Overlapping Functionality | The Operations within the realized interface of the service interface with overlapping functionality to operations of other services represent the weak point. |

According to this table, within our scenario, especially the name attributes of the different artifacts are marked as design flaws for they are responsible for the insufficient naming. In order to remove these weak points, action alternatives have to be identified in form of design decisions that are applicable during a revision and enable the improvement of the derived service designs. For service designs the following design decisions can be identified. They are again derived from existing design processes [3, 4, 5, 6, 7] and adapted for a revision of existing service designs.

TABLE VI. DESIGN DECISIONS DURING THE SPECIFICATION PHASE

| Design Decision | Description |
|---|---|
| Moving an Operation | Similar to the design decision during the identification phase, the IT architect has to decide whether or not to move an operation from one interface that is realized by a service interface into an interface realized by another service interface. |
| Renaming a Service Interface | Especially for influencing the discoverability, the IT architect can rename a service interface, i.e. the name attribute is changed. In this case, concrete action alternatives cannot be identified for the set of possible renamings is unlimited. |
| Renaming a Role | Similarly to the decision before, this design decision influences the name attribute of a role within a service interface. |
| Renaming an Operation | Whilst the naming of operation candidates was not of interest, the naming of the operations directly influences the discoverability. This design decision changes the name attribute of an operation. |
| Renaming a Parameter | This design decision changes the name attribute of a parameter that is used within an operation. |
| Renaming a Data Type | The IT architect has to decide, whether or not to rename a data type in order to increase the understanding and thus the discoverability. |
| Changing the Communication Mode of an Operation | The communication mode of an operation within an interaction protocol determines, whether the operation can be called asynchronously or not. The IT architect can change this communication mode subsequently. |
| Changing a Data Type | The data types represent information that can be used within parameters of operations. These data types can be changed. |
| Changing Parameter Types of an Operation | The parameter types of an operation represent the information that is exchanged between a service consumer and a service provider when a certain operation is called. The IT architect can change this amount and kind of information. |

In our scenario, especially the renaming of the operations and of the service interfaces are identified as action alternatives for they affect the name attributes of these artifacts that have been identified as weak points. Finally, the revised service designs can be created. Additionally, during the revision, further details of the data types can be added, such as detailed attributes. The following figure shows the revised service interface for the student administration and an excerpt of the used data types. The used KML data type represents the Keyhole Markup Language (KML) [43] that has been developed by Google for Google Earth. In the meanwhile, KML is a wide-spread markup language for geological data that has been standardized by the Open Geospatial Consortium (OGC). The other service designs are analyzed and revised equivalently. Also in these cases, mostly the names of the artifacts are changed and details are added to the data types for the service designs were derived from already revised service candidates.

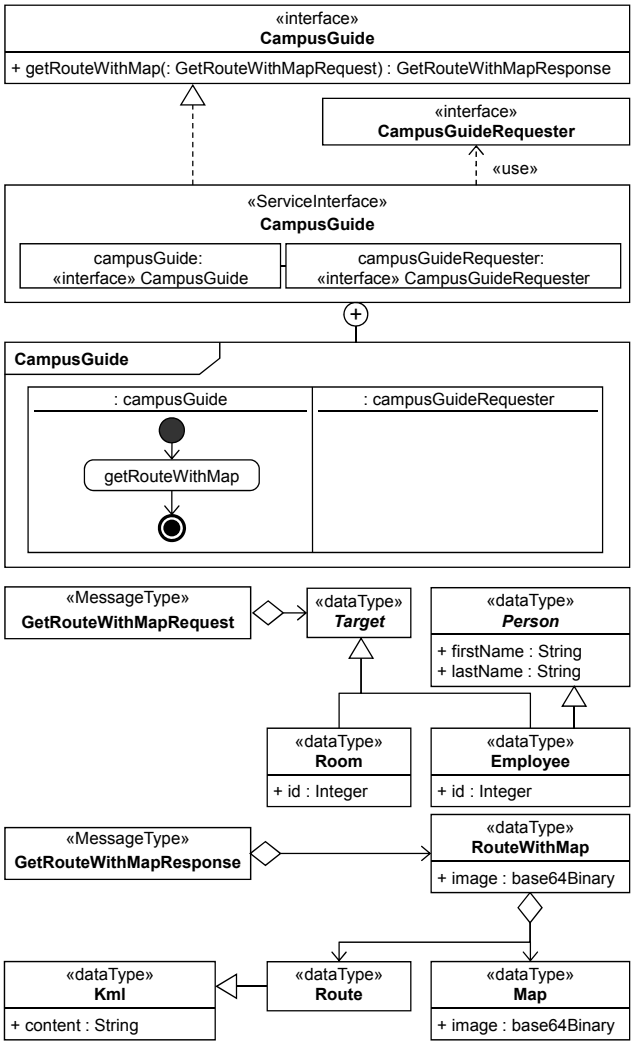


Figure 17. Revised service interface

The resulting service component for the Student Administration, now named Campus Guide, is shown below. The ServicePoints and RequestPoints have been renamed after the newly named service interfaces. The internal behavior, described as UML Activity, includes one partition for each ServicePoint and RequestPoint. Each partition that represents a RequestPoint contains CallOperationActions for the operations provided by this external service. Also in this case, the newly named operation names are used. Within the partition that represents the ServicePoint, two OpaqueActions are included. They represent internal behavior that is performed by the service component itself and is not called by external services. Since the OpaqueActions were not part of the revision, they are still named after the activities within the business process. The IT architect has to decide whether to rename these OpaqueActions. However, since they were not identified as weak points, their naming does not influence one of the considered quality attributes. Thus, a renaming would only increase the consistency within the design artifacts.

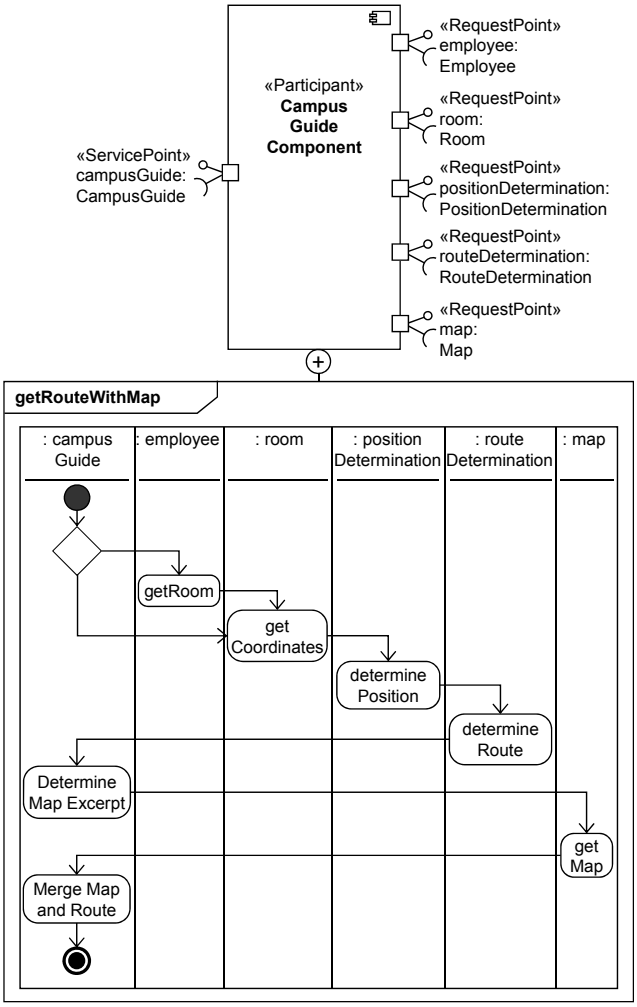


Figure 18. Revised service component

## D. *Recursive Continuation*

Till now, the design process focused on the interaction between various pools. In a next step, the activities within a pool are considered in order to increase the flexibility of the service components and their implementation. This means that the created service components are further decomposed into internal service components by recursive continuing the design process. This enables that functionality within the service components can be easily provided for external consumers or can easily be replaced by functionality that is provided by external service providers.

For this purpose, instead of the invoked activities and interaction between pools, the activities within one pool are considered and with these activities the design process is performed equivalently. First all activities within one pool are collected within one service candidate that can be named after the pool with the suffix Internal. Afterwards, the service candidate is revised according to the quality attributes and their quality indicators as described above. For our scenario, the following figure shows the derived and revised internal service candidates.

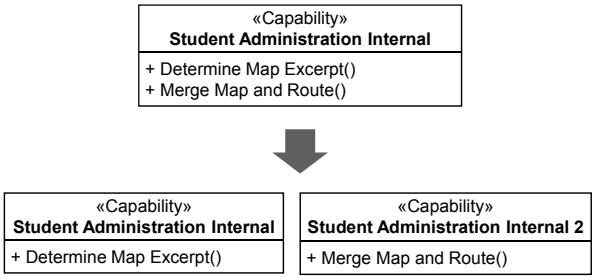


Figure 19. Internal service candidates

Afterwards, for each internal service candidate a service design is created and revised. The resulting service components are assigned to the superior service component. Since within a business process internal and external functionality is composed, a composition component is added to the superior service component. Within SoaML these internal service components are connected using the ServiceChannel element. A ServiceChannel can either be a delegation of an external ServicePoint to an internal ServicePoint respectively an internal RequestPoint to an external RequestPoint, or an assembly of two internal service components by connecting one ServicePoint with one RequestPoint. The revised service component for the Student Administration with its internal service component is depicted in Figure 20. Since there is no further decomposition necessary, the design process ends with this recursive continuation. As result, service designs are created that support the business requirements and fulfill certain quality attributes.

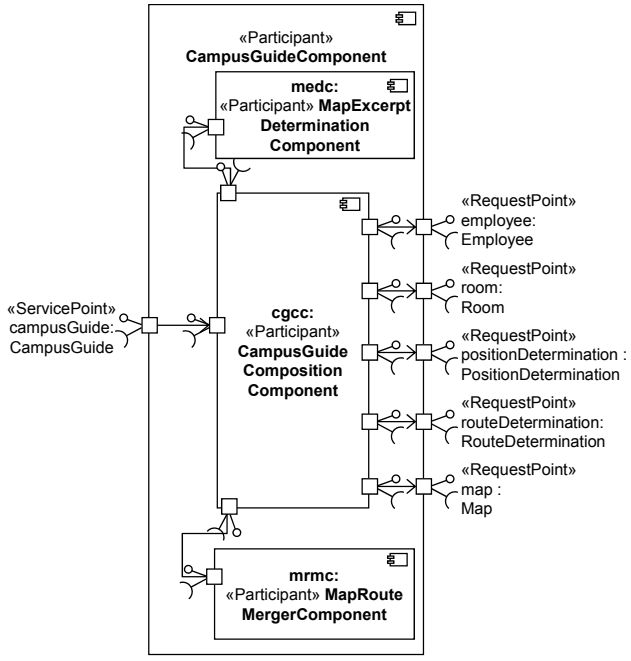


Figure 20. Internal service candidates

## IV. Conclusion and Outlook

In this article, we presented an approach for a quality-oriented design of services. The approach enhances existing design processes with a detailed description about how to transfer artifacts of the business analysis phase into service candidates and how to transfer these service candidates into service designs. Additionally, an iterative analysis and revision phase ensures the fulfillment of certain quality attributes. Due to the subsequent analysis and revision, our approach can be used in combination with other design processes and allows also the revision of already existing service designs. Due to the subsequent recursive continuation of the design process, one of the frequent questions when to use pools and when to use lanes when modeling business processes with BPMN is also solved. The recursive continuation results in the same service designs regardless of whether pools or lanes have been used. The service designs support the business requirements and fulfill a desired set of quality attributes.

The detailed description of transformations of artifacts enables the IT architect to comprehensibly derive service designs from prior created artifacts of the business analysis. Additionally, instead of only naming important quality attributes, the design process also helps to ensure their fulfillment. The usage of SoaML as language to model service candidates and service designs enables the integration of our approach into existing tool chains. SoaML represents an emerging standard for modeling service-oriented architectures. Its availability as XMI [42] enables the usage in any UML-capable development tools, although some vendors already provide built-in SoaML support.

To illustrate our approach, services of a service-oriented campus guide system as it is developed at the Karlsruhe Institute of Technology (KIT), the KITCampusGuide, have been designed. The services for this scenario could be derived comprehensibly and fulfill verifiably the quality attributes of a unique categorization, loose coupling, discoverability and autonomy. The system has its origin in the Network Enabled Surveillance and Tracking (NEST) system, developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [21, 22]. Currently, the approach is also applied for the domain campus management in order to create a catalog of services for universities and their administrative processes. These services follow national and international specifications that came up with the Bologna Process [31]. Additionally, the approach is applied at the Personalized Environmental Service Configuration and Delivery Orchestration (PESCaDO) project [35, 36], a project co-funded by the European Commission, in order to design the required services with verifiably fulfilled quality attributes.

In parallel to this article, we work on a formalization of the quality attributes and their quality indicators. Our goal is to improve our guidelines for IT architects so that the quality indicators can be measured exactly, either manually or partially even automatically. The automatically evaluable quality indicators are then formalized using the Object Constraint Language (OCL) [39] in order to enable

integration into existing development tools and thus realize a tool support for the quality-oriented design of services. Finally, we work on derivation rules to transfer design attributes into implementation artifacts [21] using technologies, such as the Service Component Architecture (SCA) [40] and the Business Process Execution Language (BPEL) [41]. While in both cases, already a lot of good work has been published, verification and if necessary an adaptation for SoaML and the semantic of service designs is required. This enables the integration of the quality-oriented design process into an entire development process.

## REFERENCES


[1] M. Gebhart, M. Baumgartner, and S. Abeck, "Supporting service design decisions", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 76-81.

[2] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck, "Evaluation of service designs based on soaml", Fifth International Conference on Software Engineering Advances (ICSEA 2010), Nice, France, August 2010, pp. 7-13.

[3] T. Erl, Service-Oriented Architecture – Concepts, Technology, and Design, Pearson Education, 2006. ISBN 0-13-185858-0.

[4] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J.-P. Richter, M. Voß, and J. Willkomm, Quasar Enteprise, dpunkt.verlag, 2008. ISBN 978-3-89864-506-5.

[5] IBM, "RUP for service-oriented modeling and architecture", IBM Developer Works, http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/, 2006. [accessed: January 04, 2011]

[6] U. Wahli, L. Ackerman, A. Di Bari, G. Hodgkinson, A. Kesterton, L. Olson, and B. Portier, "Building soa solutions using the rational sdp", IBM Redbook, 2007.

[7] A. Arsanjani, "Service-oriented modeling and architecture – how to identify, specify, and realize services for your soa", IBM Developer Works, http://www.ibm.com/developerworks/library/ws-soa-design1, 2004. [accessed: January 04, 2011]

[8] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An architectural framework for service definition and realization", 2006.

[9] T. Erl, SOA – Principles of Service Design, Prentice Hall, 2008. ISBN 978-0-13-234482-1.

[10] R. Reussner and W. Hasselbring, Handbuch der Software-Architektur, dpunkt.verlag, 2006. ISBN 978-3898643726.

[11] N. Josuttis, SOA in der Praxis – System-Design für verteilte Geschäftsprozesse, dpunkt.verlag, 2008. ISBN 978-3898644761.

[12] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, and T. Winterberg, „Die soa-service-kategorienmatrix", SOA-Spezial, Software & Support Verlag, 2009.

[13] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, and T. Winterberg, „Was macht einen guten public service aus?", SOA-Spezial, Software & Support Verlag, 2009.

[14] M. Perepletchikov, C. Ryan, K. Frampton, and H. Schmidt, "Formalising service-oriented design", Journal of Software, Volume 3, February 2008.

[15] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-Oriented design", Australian Software Engineering Conference (ASWEC 2007), 2007.

[16] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, "A metrics suite for evaluating flexibility and complexity in service oriented architecture", ICSOC 2008, 2008.

[17] S. W. Choi and S. D. Kimi, "A quality model for evaluating reusability of services in soa", 10th IEEE Conference on E-Commerce Technology and the Fifth Conference on Enterprise Computing, E-Commerce and E-Services, 2008.

[18] OMG, "Service oriented architecture modeling language (SoaML) – specification for the uml profile and metamodel for services (UPMS)", Version 1.0 Beta 1, 2009.

[19] P. Kroll and P. Kruchten, The Rational Unified Process Made Easy, a Practitioner's Guide to the RUP, Addison-Wesley, 2003.

[20] M. Gebhart and S. Abeck, "Rule-based service modeling", The Fourth International Conference on Software Engineering Advances (ICSEA 2009), Porto, Portugal, September 2009, pp. 271-276.

[21] P. Hoyer, M. Gebhart, I. Pansa, S. Link, A. Dikanski, and S. Abeck, "A model-driven development approach for service-oriented integration scenarios", 2009.

[22] T. Erl, SOA – Design Patterns, Prentice Hall, 2008. ISBN 978-0-13-613516-6.

[23] T. Erl, Web Service Contract Design & Versioning for SOA, Prentice Hall, 2008. ISBN 978-0-13-613517-3.

[24] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA – Service-Oriented Architecture Best Practices, 2005. ISBN 0-13-146575-9.

[25] S. Johnston, "UML 2.0 profile for software services", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: January 04, 2011]

[26] J. Amsden, "Modeling with soaml, the service-oriented architecture modeling language – part 1 – service identification", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html, 2010. [accessed: January 04, 2011]

[27] S. Johnston, "Rational uml profile for business modeling", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/5167.html, 2004. [accessed: January 04, 2011]

[28] J. Heumann, "Introduction to business modeling using the unified modeling language (UML)", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/360.html, 2003. [accessed: January 04, 2011]

[29] OMG, "Business process model and notation (BPMN)", Version 2.0 Beta 1, 2009.

[30] W3C, "OWL 2 web ontology language (OWL)", W3C Recommendation, 2009.

[31] European Commission, "The bologna process - towards the european higher education area", http://ec.europa.eu/education/higher-education/doc1290_en.htm, 2010. [accessed: January 04, 2011]

[32] M. Gebhart, J. Moßgraber, T. Usländer, and S. Abeck, „SoaML-basierter entwurf eines dienstorientierten beobachtungssystems", GI Informatik 2010, Leipzig, Germany, October 2010, pp. 360-367.

[33] A. Bauer, S. Eckel, T. Emter, A. Laubenheimer, E. Monari, J. Moßgraber, and F. Reinert, "N.E.S.T. – network enabled surveillance and tracking", Future Security 3rd Security Research Conference Karlsruhe, 2008.

[34] J. Moßgraber, F. Reinert, and H. Vagts, "An architecture for a task-oriented surveillance system", 2009.

[35] The PESCaDO Consortium, "Service-based infrastructure for user-oriented environmental information delivery", EnviroInfo, 2010.

[36] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, "D8.3 Specification of the pescado architecture", Version 1.0, 2010.

[37] M. Horridge, "A practical guide to building owl ontologies using protégé 4 and co-ode tools", http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/, Version 1.2, 2009. [accessed: January 04, 2011]

[38] OMG, "Unified modeling language (UML), superstructure", Version 2.2, 2009.

[39] OMG, "Object constraint language (OCL)", Version 2.2, 2010.

[40] Open SOA (OSOA), "Service component architecture (SCA), sca assembly model V1.00", http://osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf, 2009. [accessed: January 04, 2011]

[41] OASIS, "Web services business process execution language (WSBPEL)", Version 2.0, 2007.

[42] OMG, "XML metadata interchange (XMI) specification", Version 2.0, 2003.

[43] OGC, "Keyhole markup language (KML)", http://www.opengeospatial.org/standards/kml/, Version 2.2, 2008. [accessed: January 04, 2011]