# Metrics for Evaluating Service Designs Based on SoaML

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

*Abstract*—**In the context of service-oriented architectures, quality attributes, such as loose coupling and autonomy, have been identified that services should fulfill. In order to influence services with regard to these quality attributes, an evaluation is necessary at an early development stage, i.e. during design time. Existing work mostly focuses on a textual description of desired quality attributes, formalizes metrics that require more information than available during design time, or bases on a theoretical model that hampers the practical applicability. In this article, quality indicators for a unique categorization, loose coupling, discoverability and autonomy are identified. For each quality indicator formalized metrics are provided which enable their measurement and application on service candidates and service designs based on the Service oriented architecture Modeling Language as standardized language for modeling service-oriented architectures. To illustrate the metrics and to verify their validity, service candidates and service designs of a campus guide system as developed at the Karlsruhe Institute of Technology are evaluated.**

***Keywords-service design; soaml; evaluation; metric; quality attribute; quality indicator***

## I. Introduction

With the shift to service-oriented architectures, goals concerning the information technology (IT) of companies, such as an increased flexibility, are expected to be attained. In order to support this attainment, quality attributes have been identified, services within a service-oriented architecture as building-blocks [28] should fulfill. Widespread quality attributes are loose coupling, unique categorization, discoverability, and autonomy [3, 6, 8, 11, 15, 28].

Since the design of services heavily influences the services and thus their quality attributes, it is necessary to perform the design phase with care. The quality attributes of services have to be determined during design time and based on this evaluation the service designs have to be revised if necessary. For this purpose, the quality attributes have to be described in a way that the IT architect can comprehensibly apply them on service designs. This requires a formalization of the quality attributes and additionally an involvement of a standardized language for service designs that can be used in real-world projects. The Service oriented architecture Modeling Language (SoaML) [20] has evolved as increasingly accepted and employed language to model service-oriented architectures, respectively their elements. Thus, we claim quality attributes being measureable on service designs based on SoaML or comparable languages. This enables their application without additional interpretation or transformation effort.

In existing work either a textual description of quality attributes or the formalization of metrics that measure certain aspects is focused. Textual descriptions are introduced by Erl [3], Reussner et al. [6], Josuttis [7], Engels et al. [8], Cohen [10], and Maier et al. [11, 12, 13]. They introduce a comprehensive set of quality attributes that should be considered when developing services. However, due to the textual descriptions, an application of these quality attributes is hampered and requires a prior interpretation. Each of the described quality attributes covers a lot of different aspects. Some of these aspects are already relevant during design time and others are only of interest in subsequent phases, such as the implementation phase. The IT architect has to analyze the quality attributes and identify relevant and measurable quality indicators first. Then he has to interpret them so that they can be applied on a modeled service design. Other work, as introduced by Perepletchikov et al. [14, 15, 16], Humm et al. [9], Rud et al. [17], Hirzilla et al. [18], and Choi et al. [19] focus on the formalization of metrics. Some of these formalizations base on theoretical models. This hampers the application on service designs that have been modeled using a standardized and widespread modeling language, such as SoaML, for a prior mapping of the different concepts is required. Additionally, the metrics are mostly abstractly and conceptually described which requires a prior interpretation again. For example concepts, such as “number of clients”, are used and the IT architect has to interpret if services or operations are meant and if duplicates should be counted or not. This interpretation may result in mistakes and consequently wrong evaluations. Finally, the metrics are often not related to widespread quality attributes that support the attainment of goals concerning the IT. Metrics that are not related to these widespread quality attributes are not motivated.

In this article, we derive quality indicators for the quality attributes unique categorization, loose coupling, discoverability, and autonomy. These quality attributes were chosen for they contain a representative set of aspects that can be measured during design time as shown in [1]. The quality indicators are derived directly from common and widespread descriptions of quality attributes in order to preserve the relation of the quality indicators to quality attributes and thus motivate their measurement. Additionally, each quality indicator is formalized in form of metrics using the notation as introduced by Perepletchikov [13] in order to enable a comprehensible measurement. The formalization is adjusted to the elements of service candidates or service designs and their elements in SoaML. This allows a direct application of the quality indicators and their formalizations without any interpretation effort and thus reduces potential interpretation mistakes.

To illustrate the metrics and their validity, they are exemplarily applied on service candidates and service designs for a service-oriented system that guides students across the campus of the Karlsruhe Institute of Technology (KIT) called KITCampusGuide. This system has its origin in the NEST system, a service-oriented surveillance system developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [37, 38, 39]. As requirement, the services of the KITCampusGuide are expected to fulfill the quality attributes unique categorization, loose coupling, discoverability, and autonomy. The service candidates as preliminary services and the final service designs are modeled using SoaML.

The article is organized as follows: In Section 2, the fundamentals in the context of modeling service candidates and service designs using SoaML and the evaluation of services are presented. Section 3 introduces the derived quality indicators and their formalizations. The quality indicators are exemplarily applied on service candidates and service designs of the service-oriented KITCampusGuide. Section 4 concludes the article and introduces suggestions for future research.

## II. Fundamentals

This article focuses on the evaluation of service designs. This requires a prior understanding of the concept of a service design that is introduced in the following section. Afterwards existing work in the context of evaluating services is analyzed with regard to their application on service designs. Finally, the existing work is discussed which constitutes the motivation for this article.

### *A. Modeling Service Designs*

In order to analyze existing work with regard to their applicability for evaluating service designs, in the following, the service design artifact and its elements in SoaML are introduced. According to Erl [4, 5] and the Rational Unified Process (RUP) [24] for Service-Oriented Modeling and Architecture (SOMA) [21, 22, 23], the design process consists of two phases, the identification phase and the specification phase.

*1) Identification Phase:* The first phase, the identification phase, focuses on the determination of so-called service candidates [25]. They represent preliminary services as an abstract group of capabilities the service provides. A service candidate includes operation candidates as capabilities of the service candidate and preliminary operations. Additionally, the dependencies between service candidates are modeled. They describe that an operation candidate within a certain service candidate requires an operation candidate of another service candidate. In SoaML, service candidates are modeled using the Capability element in form of a stereotyped UML class. The operation candidates are added as UML operations. Dependencies can be modeled with usage dependencies in UML. The application of the Capability elements for service candidates is also confirmed by IBM. Within RUP SOMA, IBM uses its proprietary UML profile for modeling service candidates, the UML 2.0 profile for software services [33]. However, in newest work [34] they demonstrate the modeling of service candidates on the basis of SoaML which correlates with our understanding. The following Figure illustrates a set of service candidates and their dependencies.

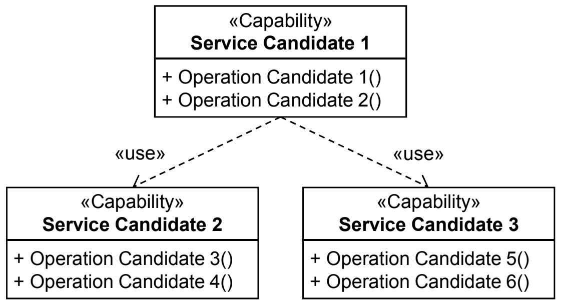
«Capability»
Service Candidate 1
+ Operation Candidate 1()
+ Operation Candidate 2()
«use»
«use»
«Capability»
Service Candidate 2
+ Operation Candidate 3()
+ Operation Candidate 4()
«Capability»
Service Candidate 3
+ Operation Candidate 5()
+ Operation Candidate 6()

Figure 1. Modeling service candidates

*2) Specification Phase:* During the second phase, the specification phase, for each of the service candidates that constitute the basis for the implementation phase detailed service designs are created [26]. According to Erl [4, 27] and IBM [21, 22], a service design consists of a specified service interface that describes the service and a specification of the service component that fulfills the functionality. Latter includes the provided and required services and the internal logic in form of an orchestration of required services.

For modeling a service interface in SoaML the ServiceInterface element is provided that can be modeled using a stereotyped UML class. It includes a description about participating roles as UML Parts, realizes a UML interface that contains the provided operations, and uses another UML interface that includes operations a service consumer has to provide in order to receive callbacks. Additionally, the interaction protocol can be specified that describes the order of operations for gaining a valid result. The interaction protocol can be described using a UML Activity that is added as OwnedBehavior. Within this Activity, for each participating role a Partition is added containing the operations of the according UML interfaces. Figure 2 shows a service interface in SoaML. According to this service interface, the described service provides one operation called operation1. The service consumer has to provide one operation callbackOperation1 in order to receive a callback. The interacting roles are referred to as provider and consumer. The interaction protocol determines that for a valid result first the operation1 has to be called that is provided by the service. Afterwards, the callbackOperation1 of the service consumer is called by the service provider.

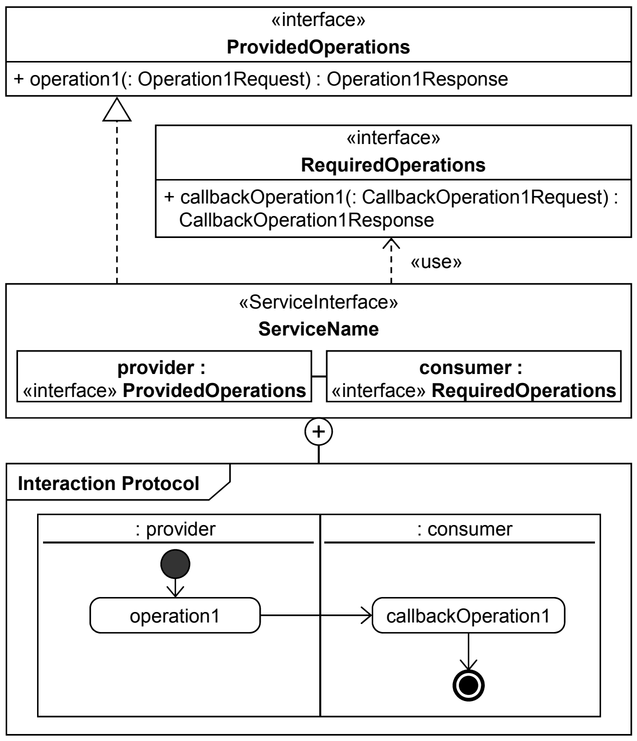
«interface»
ProvidedOperations
+ operation1(: Operation1Request) : Operation1Response
«interface»
RequiredOperations
+ callbackOperation1(: CallbackOperation1Request) : CallbackOperation1Response
«use»
«ServiceInterface»
ServiceName
provider : «interface» ProvidedOperations
consumer : «interface» RequiredOperations
Interaction Protocol
: provider
: consumer
operation1
callbackOperation1

Figure 2. Modeling a service interface

Since each of the operations within the interfaces include messages being exchanged, a specification of these messages is necessary. For this purpose, the MessageType element is provided by SoaML that extends the UML data Type. It represents a document-centric message and can contain other data types. The following figure shows an excerpt of the message types required within the service interface above.

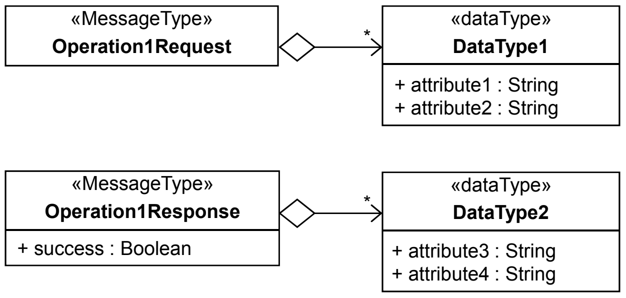

Figure 3. Modeling message types

The service component as part of the service design describes the component that fulfills the functionality of a service. For this purpose in SoaML the Participant element exists. It describes an organization, system, or software component. The service component is modeled using a stereotyped UML component. For each provided service, a ServicePoint is added to the service component that is typed by the describing ServiceInterface. Similarly, for each required service, a RequestPoint is specified that is also typed by the according ServiceInterface. The following figure illustrates a service component in SoaML.

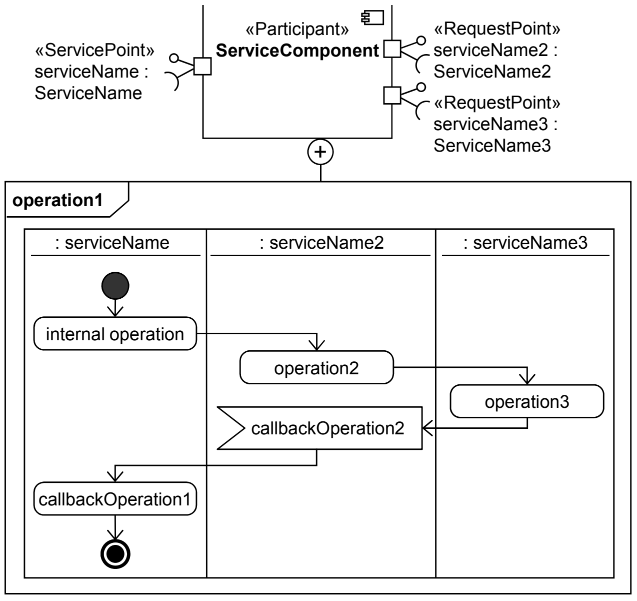

Figure 4. Modeling a service component

The internal logic is modeled using one UML Activity for each operation that is performed by the service component. The Activity is added as OwnedBehavior and named after this operation. It contains one UML Partition for each ServicePoint and RequestPoint. If an operation of a required service is called, a CallOperationAction is added to the according Partition. For receiving callbacks an AcceptEvent is used. If the service component performs functionality by itself, i.e. functionality that is not provided by an external service, an OpaqueAction is added to the Partition that represents the ServicePoint.

### B. *Evaluating Services*

According to Boehm [29] and McCall et al. [30], within the Factor Criteria Metrics (FCM) quality model, the software quality as factor can be broken down into several criteria that can be further be described by metrics. When considering the flexibility, maintainability etc. as factor, the quality attributes considered within this article, such as loose coupling, can be thought of as criteria. Since the criteria are not measurable, they have to be further refined into metrics that equal quality indicators as introduced by the International Organization for Standardization (ISO) [31]. Thus, in order to evaluate service designs with regard to quality attributes that have been identified as criteria for services of high quality, quality indicators have to be determined and formalized as metrics. Within existing work either a description of quality attributes or metrics that enable the measurement of quality indicators are focused.

*1) Description of Quality Attributes:* In [3, 4], Erl introduces a comprehensive set of design principles and design patterns for services that can be thought of as quality attributes. The design principles are described in detail, however only textual information is provided. Quality indicators that enable an evaluation of service designs with regard to quality attributes are not introduced. Only some so-called service characteristics are described that represent the impact of the design principles. However, these service characteristics are mostly not directly measurable and an explanation how to evaluate service designs with regard to these service characteristics is missing. Also some of these characteristics can only be evaluated if implementation details and deployment information are available.

For the Rational Unified Process for Service Oriented Modeling and Architecture (RUP SOMA), IBM lists quality attributes that have to be considered [21, 22, 23]. However, the quality attributes are only listed and considered as important, but a detailed description about these attributes and how to measure them is not provided likewise.

Engels et al. describe in [8] a method to design an application landscape and focus on the development process. Necessary steps to derive services from a prior analyzed business are explained and during the design phase several quality attributes are named. Also in this case, the quality attributes are only described textually. Quality indicators or metrics are not included. Similarly, Reussner et al. [6], Josuttis [7] and Maiers et al. [11, 12, 13] introduce quality attributes for services within service-oriented architectures. Also in this work, the textual description is focused and quality indicators or metrics are missing.

*2) Formalized Metrics:* Other work focuses on the formalization of metrics to evaluate services. Perepletchikov et al. [14, 15, 16] introduce metrics to measure cohesion and coupling of services. The metrics base on an extension of the generic software model of Briand [32]. Rud et al. [17] show metrics for measuring the granularity of services and Hirzialla et al. [18] focus on the flexibility. Choi et al. [19] measure the reusability of services. All this work has in common that it is not mainly meant for evaluating service designs as introduced in the section before. In some cases information about the implementation of the service or its deployment is required. The metrics that concentrate on the design of services and the information available during this time base on own notations and own understandings about how to design a service. For using the metrics on a common and standardized language, such as SoaML, first the used concepts within the metrics have to be transferred into representations within the formalized service designs. For this, a prior interpretation of the metric and a detailed understanding about the used concepts is necessary in order to correctly apply the metric. Additionally, some of the introduced metrics are not related to common and widespread quality attributes which hampers their incentive. to measure them.

### *C. Discussion*

As illustrated above, SoaML provides all necessary elements to model service designs. However, SoaML does not provide any information about how to design services with certain quality attributes. On the other side, work in the context of quality attributes and formalized metrics is mostly not directly applicable due to the fact that the textual descriptions are too abstract. Furthermore, the formalized metrics are often not related to widespread quality attributes and they use concepts and understandings of service designs that have to be mapped onto elements of concrete languages first. This step requires an interpretation of the metrics and may include interpretation mistakes. Thus, in this article the quality attributes and their textual descriptions are used for deriving quality indicators and formalized metrics that can be applied on a formalized service design based on SoaML. The metrics are described by means of a similar notation as introduced by Perepletchikov et al. [14]. Due to the direct derivation of quality indicators from quality attributes and the usage of SoaML, the motivation why to measure them is given and a direct application without any interpretation effort is enabled.

## III. METRICS FOR EVALUATING SERVICE DESIGNS BASED ON SOAML

In order to evaluate service designs based on SoaML, quality indicators have to be identified that give the IT architect hints about the current value of the quality attributes. In this article, the four quality attributes of a unique categorization, loose coupling, discoverability, and autonomy are considered. These quality attributes mostly require information that is available during design time compared to quality attributes, such as statelessness and idempotence, which require additional information [1].

To illustrate the quality indicators, in this article the human-centered environmental observation domain is considered. This domain refers to the network-enabled surveillance and tracking system as introduced by the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [38, 39]. Currently, at the Karlsruhe Institute of Technology (KIT) the KITCampusGuide, a system to provide a guide for students, lecturers and guest, is developed. A person can ask for another person or a room on the campus of the university using mobile devices, such as a mobile phone, and the KITCampusGuide calculates the route. The following figure shows the KITCampusGuide in action. A requirement for the KITCampusGuide is to create services for this scenario that fulfill the quality attributes of a unique categorization, loose coupling, autonomy and discoverability as introduced in [2]. This is why this scenario is chosen to illustrate the quality indicators and their formalizations.

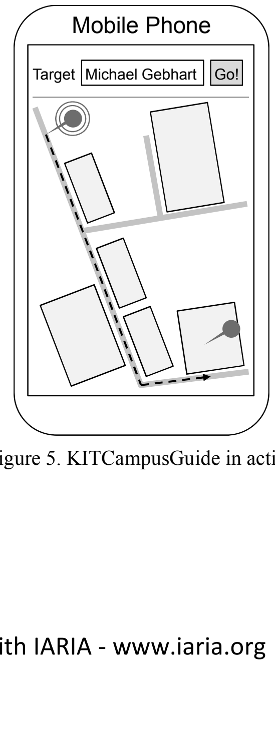

Figure 5. KITCampusGuide in action

Since some of the quality indicators require knowledge about the functional terms used within this scenario, the domain is modeled using an ontology based on the Web Ontology Language (OWL) [35]. As modeling tool Protégé [42] is applied. For illustrating the ontology we choose a notation that is similar to the OntoGraf in Protégé. Each concept is depicted by a rectangle and the relations between these concepts are represented by lines between these rectangles. In order to provide the name of the concepts and relations in various languages, the translations are added as labels. A suffix specifies the language, such as "@de" for German. The following figure shows an excerpt of the domain model for the human-centered environmental observation.

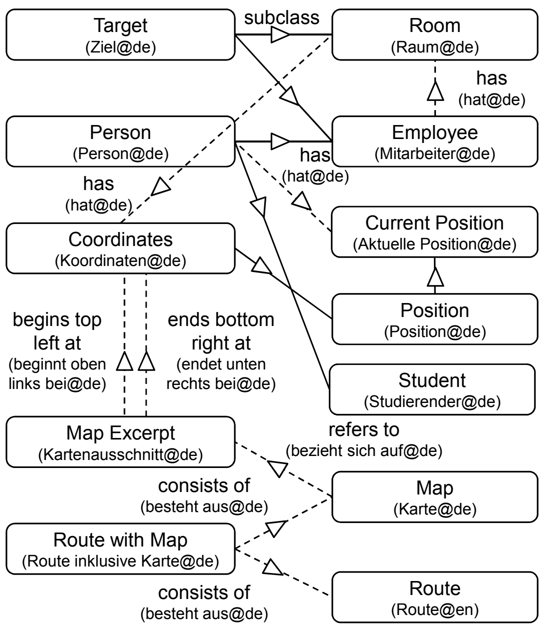

Figure 6. Excerpt of the domain model

For identifying quality indicators, the descriptions of the quality attributes are analyzed and aspects that can be measured within a service design are extracted. Each of these aspects represents a quality indicator. Afterwards, for each quality indicator a formalized metric is created. This formalization uses elements of a service design which enables its direct applicability. To interpret the value of the metric correctly, a scale is assigned. An exemplarily application of the metric on a service design of our scenario helps to illustrate the metric and verifies its validity.

## *A. Unique Categorization*

Erl [3], Cohen [10], Engels et al. [8], and Maier et al. [12] introduce approaches for a unique categorization. The categorization has its origin in splitting different and bundling similar kinds of functionality. Thus, the unique categorization corresponds to the concept of cohesion [3]. Common categories are entity services that are responsible for managing business entities, task services that provide mostly process-specific functionality beyond the scope of one business entity, and utility services for cross-cutting technical functionality. This categorization can be broken down into different aspects that represent quality indicators.

*1) Division of Business-Related and Technical Functionality:* According to Reussner et al. [6], functionality that changes in different time intervals, such as business-related and technical functionality, should be split into several modules, in this case services because this increases their maintainability. Business-related functionality refers to the logic of the business domain, whilst technical functionality includes cross-cutting technical functionality, as for instance functionality of logging systems or security systems. The division of these different kinds of functionality categorizes services into entity and task services on the one side and utility services on the other side.

On the basis of service candidates the division can be verified by means of the contained operation candidates. Appropriate knowledge about the purpose of these operation candidates assumed, the following metric can be formalized.

$$DBTF(sc) = \frac{|\,BF(OC(sc))\,|}{|\,OC(sc)\,|}$$

In case of service designs, instead of the operation candidates the operations of the realized interface are considered.

$$DBTF(s) = \frac{\left|BF\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|}$$

The metrics are only valid if there exists any operation candidate respectively operation. Within the metrics, the following variables and functions are used.

TABLE I. VARIABLES AND FUNCTIONS USED FOR DBTF

| Element | Description |
|---|---|
| DBTF | Division of Business-related and Technical Functionality |
| sc | service candidate: the considered service candidate |
| s | service: the considered service that is provided or required, represented by an ServicePoint or RequestPoint in SoaML |
| BF(oc) | Business Functionality: operation candidates providing business-related functionality out of the set of operation candidates oc |
| BF(o) | Business Functionality: operations providing business-related functionality out of the set of operations o |
| OC(sc) | Operation Candidates: operation candidates of the service candidate sc |
| SI(s) | Service Interface: service interface of the service s. In SoaML it is the type of the ServicePoint or RequestPoint s. |
| RI(si) | Realized Interfaces: realized interfaces of the service interface si |
| O(i) | Operations: operations within the interface i |
| \| oc \| | Number of operation candidates oc |
| \| o \| | Number of operations o |

Thus, the metrics return values from 0 to 1, which have an order. Accordingly, the results are interpreted within the ordinal scale. The following table shows the interpretation of values for DBTF.

TABLE II. INTERPRETATION OF VALUES FOR DBTF

| Value | Interpretation |
|---|---|
| 0 | Only technical functionality is provided |
| Between 0 and 1 | Both business-related and technical functionality is provided |
| 1 | Only business-related functionality is provided |

According to this table, for DBTF a value of 0 or 1 is desired because these values represent a division of business-related and technical functionality. A value between 0 and 1 should be avoided. The following figure depicts one service candidate and one interface realized by a service interface. The former includes both operation candidates with business-related and technical functionality. The latter includes only business-related functionality. Thus, the design of the service candidate should be revised, whilst the design of the interface follows the criteria for a service that can be uniquely categorized. This circumstance is also confirmed by the metric DBTF. For the service candidate a value of 0.5 and for the service interface a value of 1 is returned.

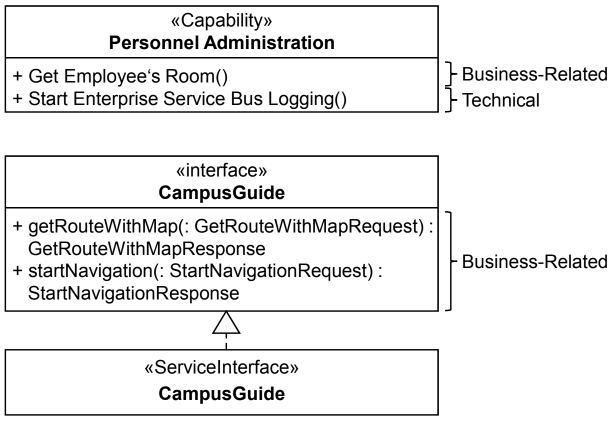
«Capability»
Personnel Administration
+ Get Employee's Room()
+ Start Enterprise Service Bus Logging()
Business-Related
Technical
«interface»
CampusGuide
+ getRouteWithMap(: GetRouteWithMapRequest) : GetRouteWithMapResponse
+ startNavigation(: StartNavigationRequest) : StartNavigationResponse
Business-Related
«ServiceInterface»
CampusGuide

Figure 7. Example for division of business-related and technical functionality

2) *Division of Agnostic and Non-agnostic Functionality:* In order to increase the reusability of services, agnostic functionality that is agnostic should be divided from non-agnostic functionality [3]. Agnostic functionality is highly reusable and not process-specific, whilst non-agnostic functionality is less reusable and mostly process-specific. A usage of non-agonstic functionility in several processes is not expected. The division of this functionality results in the distinction of agnostic services, such as entity services, and non-agnostic services, such as task services.

Similarly to the division of business-related and technical functionality, on the basis of service candidates the division of agnostic and non-agnostic functionality can be evaluated by considering the operation candidates. Equivalently to DBTF, the following metric measures the ratio of operation candidates providing agnostic functionality to all operation candidates.

$$DANF(sc) = \frac{|AF(OC(sc))|}{|OC(sc)|}$$

If service designs are supposed to be evaluated, instead of the operation candidates the operations of the realized interface are used within the metric.

$$DANF(s) = \frac{\left|AF\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|}$$

The metrics are only valid if there is at least one operation candidate respectively one operation provided. Variables and functions that are used additionally to those introduced for DBTF are listed below.

TABLE III. VARIABLES AND FUNCTIONS USED FOR DANF

| Element | Description |
|---|---|
| DANF | Division of Agnostic and Non-agnostic Functionality |
| AF(oc) | Agnostic Functionality: operation candidates providing agnostic functionality out of the set of operation candidates oc |
| AF(o) | Agnostic Functionality: operations providing agnostic functionality out of the set of operations o |

The metrics return values from 0 to 1. Also in this case, the results are interpreted within the ordinal scale.

TABLE IV. INTERPRETATION OF VALUES FOR DANF

| Value | Interpretation |
|---|---|
| 0 | Only non-agnostic functionality is provided |
| Between 0 and 1 | Both agnostic and non-agnostic functionality is provided |
| 1 | Only agnostic functionality is provided |

In order to increase the unique categorization, a value of 0 or 1 is desired. The following figure shows a service candidate with both agnostic and non-agnostic operation candidates. In order to increase the reusability, these operation candidates should be divided. This is also confirmed by the value 0.5 that is returned for DANF.

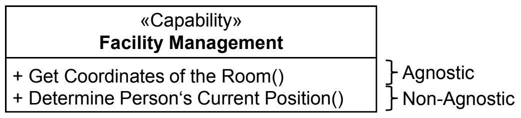
«Capability»
Facility Management
+ Get Coordinates of the Room()
+ Determine Person's Current Position()
Agnostic
Non-Agnostic

Figure 8. Example for division of agnostic and non-agnostic functionality

3) *Data Superiority:* If a service manages a certain business entity, this service should be explicitly managing this entity. For example, if a service is responsible for creating, deleting or changing a business entity, no other service should provide similar functionality. This concept is

called data superiority [8]. According to Erl [3] and Cohen [10], a service that fulfills the data superiority corresponds to an entity service.

On the basis of service candidates, the business entities managed by the contained operation candidates have to be assumed and compared with the business entities managed by operation candidates of other services. Optimally, there should be no overlap.

$$DS(sc) = 1 - \frac{|MBE(OC(sc)) \cap MBE(OC(ALL_{SC} \setminus sc))|}{|MBE(OC(sc))|}$$

On the basis of service designs the metric can be formalized as follows.

$$DS(s) = 1 - \frac{\left| \begin{matrix} MBE\left(O\left(RI(SI(s))\right)\right) \cap \\ MBE\left(O\left(RI\left(SI((ALL_S \setminus s))\right)\right)\right) \end{matrix} \right|}{\left|MBE\left(O\left(RI(SI(s))\right)\right)\right|}$$

TABLE V. VARIABLES AND FUNCTIONS USED FOR DS

| Element | Description |
|---|---|
| DS | Data Superiority |
| M1 \ M2 | Elements of set M1 without elements of set M2 or the element M2 |
| $ALL_{SC}$ | All existing service candidates |
| $ALL_S$ | All existing services |
| MBE(oc) | Managed Business Entities: business entities that are managed by operation candidates oc |
| MBE(o) | Managed Business Entities: business entities that are managed by operations o |

The metrics require that at least one business entity is managed by the service candidate respectively service. The metrics return values from 0 to 1. Based on the ordinal scale the results can be interpreted as follows.

TABLE VI. INTERPRETATION OF VALUES FOR DS

| Value | Interpretation |
|---|---|
| Less than 1 | No data superiority regarding the managed business entities |
| 1 | Data superiority regarding the managed business entities |

For the metrics a value of 1 is desired. The following figure shows three service candidates that manage business entities. Assumed that there exist no other service candidates, for the service candidate Personal Management the value of DS is 1 because it manages explicitly the business entity Employee. For the service candidates Facility Management and Room, the value of DS is 0, for both manage the same business entities. In order to increase DS, the IT architect should consider a merger of these service candidates.

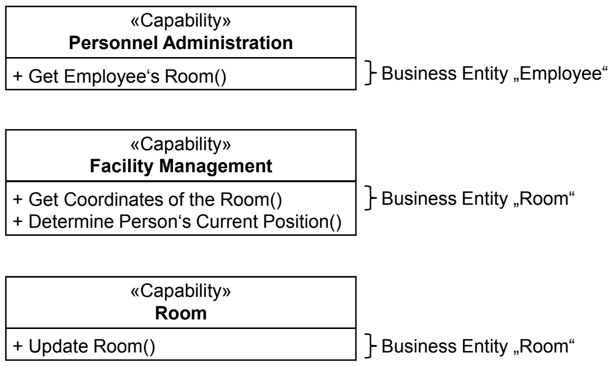

Figure 9. Example for data superiority and common business entity usage

4) *Common Business Entity Usage:* Additionally, the provided operations should use common business entities [PR+07] for ensuring that, for instance, an entity service focuses on one business entity only. This means that the business entities that are used as input parameters within operation candidates respectively operations should either be identical or should be dependent. A business entity depends from another business entity if it cannot exist for its own. This concept is comparable to the composition within UML [43].

On the basis of service candidates the common business entity usage can be evaluated using the operation candidates and their propably used business entities. First, all used business entities of the operation candidates and the within one operation candidate mostly often used business entities are determined. From these two sets of business entities the biggest set of common, i.e. depending business entities is created. Afterwards, the operation candidates that use these business entitites are identified and related to all operation candidates.

$$CBEU(sc) = \frac{\left| OCUBE\left( OC(sc), CMP\begin{pmatrix} MOUBE(OC(sc)), \\ UBE(OC(sc)) \end{pmatrix} \right) \right|}{|OC(sc)|}$$

On service designs, the operations within the realized interface instead of operation candidates are used.

$$CBEU(s) = \frac{\left| OUBE\left( \begin{matrix} O\left(RI(SI(s))\right), \\ CMP\begin{pmatrix} MOUBE\left(O\left(RI(SI(s))\right)\right), \\ UBE\left(O\left(RI(SI(s))\right)\right) \end{pmatrix} \end{matrix} \right) \right|}{\left| O\left(RI(SI(s))\right) \right|}$$

The metrics require that there exists at least one operation candidate respectively one operation. Within these metrics, the following additional variables and functions are used.

TABLE VII. VARIABLES AND FUNCTIONS USED FOR CBEU

| Element | Description |
|---|---|
| CBEU | Common Business Entity Usage |
| CMP(be1, be2) | Composition: biggest set of business entities out of be2 that depend on business entitites be1 |
| UBE(oc) | Used Business Entities: business entities that are used within operation candidates oc as input |
| UBE(o) | Used Business Entities: business entities that are used within operations o as input |
| MOUBE(oc) | Mostly Often Used Business Entities: business entities that are mostly often used within one operation candidate out of operation candidates oc |
| MOUBE(o) | Mostly Often Used Business Entities: business entities that are mostly often used within one operation out of operations o |
| OCUBE(oc, be) | Operation Candidates Using Business Entities: operation candidates out of operation candidates oc that only use business entities out of be |
| OUBE(o, be) | Operations Using Business Entities: operations out of operations o that only use business entities out of be |

The metrics return results from 0 to 1. The interpretation is listed below based on the ordinal scale.

TABLE VIII. INTERPRETATION OF VALUES FOR CBEU

| Value | Interpretation |
|---|---|
| Less than 1 | There exist operation candidates respectively operations that use non-common business entitites |
| 1 | All operation candidates respectively operations use common business entitites |

For CBEU a value of 1 is desired because this value represents the case that all operation candidates or operations use common business entities. Applied on Figure 9, the service candidate Facility Management contains one operation candidate that uses the Room entity and one operation candidate that uses the Person entity as input. Since a room can exist without a person and vice versa, the metric returns a value of 0.5. A merge of the service candidate Room with the service candidate Facility Management as already proposed to improve the data superiority quality indicator would increase the value for CBEU. An optimal value could be achieved if the two contained operation candidates would be divided into two different service candidates.

## B. Discoverability

Since reusability of existing functionality is one major aspect when establishing a service-oriented architecture, services are required to be discoverable. The discoverability is already positively influenced by a unique categorization. However, there are other aspects, such as naming conventions, which have to be considered. These aspects constitute quality indicators that influence the discoverability.

*1) Functional Naming:* The first quality indicator focuses on the functional naming of the created artifacts. According to Josuttis [7], in order to understand the functionality a service provides, the description of a service has to follow functional terms that have been determined for the considered domain. This means that the elements of a service, such as its interface, the roles, and the operations, have to be named after functional terms as they are determined within a domain model. Thus, the functional naming of a service can be further broken down into a functional naming of its externally visible artifacts, i.e. of the service interface, the roles, the operations, the parameters and the data types.

Since the service candidates are mainly meant to describe the architecture, i.e. the services and their dependencies in an abstract manner, the naming of artifacts constitutes a quality indicator only of interest on the basis of service designs. Thus, the following metrics for evaluating the functional naming are only measurable on service designs. The metrics determine the ratio of functionally named artifacts compared to all artifacts. For evaluating the functional naming of the service interface, the following metric can be formalized:

$$FNSI(s) = \frac{|FN(SI(s))|}{|SI(s)|}$$

The metrics for evaluating the functional naming of roles, operations, parameters, and data types can be formalized equivalently.

$$FNR(s) = \frac{\left|FN\left(R(SI(s))\right)\right|}{\left|R\left(SI(s)\right)\right|}$$

$$FNO(s) = \frac{\left|FN\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|}$$

$$FNP(s) = \frac{\left|FN\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}{\left|P\left(O\left(RI(SI(s))\right)\right)\right|}$$

$$FNDT(s) = \frac{\left|FN\left(DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right)\right|}{\left|DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}$$

TABLE IX. VARIABLES AND FUNCTIONS USED FOR FNSI, FNR, FNO, FNP, AND FNDT

| Element | Description |
|---|---|
| FNSI | Functional Naming of Service Interface |
| FNR | Functional Naming of Roles |
| FNO | Functional Naming of Operations |
| FNP | Functional Naming of Parameters |

| | |
|---|---|
| FNDT | Functional Naming of Data Types |
| FN(me) | Functional Naming: set of functionally named elements out of the set of modelling elements me |
| P(o) | Parameters: parameters of the operations o and in case of messages the contained parameters |
| DT(p) | Data Types: used data types (recursively continued) of parameters p |
| R(si) | Roles: roles of service interface si |

Each of the metrics is only valid if there is at least one role, one operation, one parameter, respectively one data type specified within the considered service design. Otherwise, the metric cannot be applied. As result, values from 0 to 1 are returned. The interpretation based on the ordinal scale is shown below.

TABLE X. INTERPRETATION OF VALUES FOR FNSI, FNR, FNO, FNP, AND FNDT

| Value | Interpretation |
|---|---|
| Less than 1 | There are elements that are not functionally named |
| 1 | All elements are functionally named |

For the metrics FNSI, FNR, FNO, FNP, and FNDT, a value of 1 is desired. For example, based on the domain model as depicted in Figure 6, the FNO for the CampusGuide in Figure 7 is 0.5, for the term navigation is not part of the domain model. The IT architect has to decide if the term is functional and was accidently not added to the domain model. In this case the domain model should be revised. Otherwise the name of the operation should be changed. Thus, this metric does not only evaluate service designs. It also helps IT architect to validate the prior created domain model.

2) *Naming Convention Compliance:* The second quality indicator for the discoverability addresses the compliance with naming conventions. According to Maier et al. [13], the compliance with naming conventions increases the discoverability of a service. Typical naming conventions are the usage of the english language, verbs possibly followed by nouns for the names of operations, and upper case letters at the beginning of the name of service interfaces and data types. These naming conventions are necessary to create consistently named artifacts.

Similarly to the functional naming, this quality indicator is only of interest for service designs. The quality indicator can be further broken down into a naming convention compliance of the service interface, the roles, the operations, the parameters, and the data types. The metrics are similarly formalized to the metrics for the functional naming and determine the ratio of artifacts named regarding naming conventions to all artifacts. The metrics are only defined if the particular artifacts, i.e. the service interface, the roles, provided operations, used parameters, and data types are specified. Otherwise the metrics cannot be applied on the considered service design.

$$NCCSI(s) = \frac{|NCC(SI(s))|}{|SI(s)|}$$

$$NCCR(s) = \frac{\left|NCC\left(R(SI(s))\right)\right|}{\left|R\left(SI(s)\right)\right|}$$

$$NCCO(s) = \frac{\left|NCC\left(O\left(RI(SI(s))\right)\right)\right|}{\left|O\left(RI(SI(s))\right)\right|}$$

$$NCCP(s) = \frac{\left|NCC\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}{\left|P\left(O\left(RI(SI(s))\right)\right)\right|}$$

$$NCCDT(s) = \frac{\left|NCC\left(DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right)\right|}{\left|DT\left(P\left(O\left(RI(SI(s))\right)\right)\right)\right|}$$

TABLE XI. VARIABLES AND FUNCTIONS USED FOR NCCSI, NCCR, NCCO, NCCP, AND NCCDT

| Element | Description |
|---|---|
| NCCSI | Naming Convention Compliance of Service Interface |
| NCCR | Naming Convention Compliance of Roles |
| NCCO | Naming Convention Compliance of Operations |
| NCCP | Naming Convention Compliance of Parameters |
| NCCDT | Naming Convention Compliance of Data Types |
| NCC(me) | Naming Convention Compliance: set of elements out of the set of modelling elements me that follow specified naming conventions |

The metrics return values from 0 to 1, interpreted on the basis of the ordinal scale.

TABLE XII. INTERPRETATION OF VALUES FOR NCCSI, NCCR, NCCO, NCCP, AND NCCDT

| Value | Interpretation |
|---|---|
| Less than 1 | There are elements that do not follow naming conventions |
| 1 | All elements follow naming conventions |

The following figure depicts an interface realized by a service interface and the evaluation regarding naming conventions. According to this figure and the available information, the NCCSI is 0, NCCR is 0.5, NCCO is 0.5, NCCP is 0.75, and NCCDT is 0. Thus, the IT architect should revise the service designs and especially the names of the artifacts in order to increase the discoverability of the resulting service.

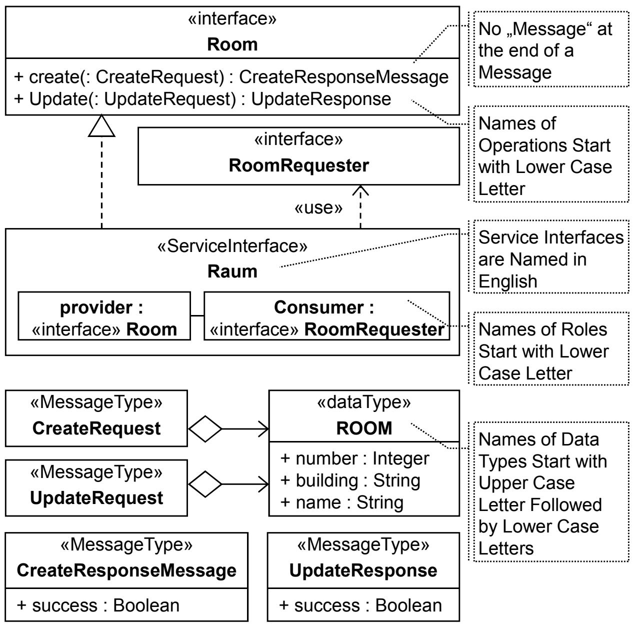
«interface»
Room
+ create(: CreateRequest) : CreateResponseMessage
+ Update(: UpdateRequest) : UpdateResponse
«interface»
RoomRequester
«use»
«ServiceInterface»
Raum
provider : «interface» Room
Consumer : «interface» RoomRequester
«MessageType»
CreateRequest
«MessageType»
UpdateRequest
«dataType»
ROOM
+ number : Integer
+ building : String
+ name : String
«MessageType»
CreateResponseMessage
+ success : Boolean
«MessageType»
UpdateResponse
+ success : Boolean
No „Message“ at the end of a Message
Names of Operations Start with Lower Case Letter
Service Interfaces are Named in English
Names of Roles Start with Lower Case Letter
Names of Data Types Start with Upper Case Letter Followed by Lower Case Letters

Figure 10. Example for naming convention compliance

*3) Information Content:* The service interface describes a service from an external point of view for potential service consumers. According to Erl [3], the extent of this information influences the discoverability of a service.

Transferred to SoaML, a service interface should contain as much information as possible. This means that the service interface, the contained roles, the realized interface, the used interface and the interaction protocol should be formalized. As metric the extent of the information content can be described as follows:

$$IC(s) = \frac{EX(SI(s)) + EX\left(RI(SI(s))\right) + EX\left(UI(SI(s))\right) + EX\left(R(SI(s))\right) + EX\left(IP(SI(s))\right)}{5}$$

TABLE XIII. VARIABLES AND FUNCTIONS USED FOR IC

| Element | Description |
|---|---|
| IC | Information Content |
| EX(e) | Exists: returns 1 if the element e exists, else 0 |
| IP(si) | Interaction Protocol: interaction protocol of the service interface si |
| UI(si) | Used Interfaces: used interface provided by the service consumer |

As result, values from 0 to 1 are returned. The interpretation based on the ordinal scale is shown in the following table. For IC a value of 1 is desired for this value represents the case that all possible information is available within the service design.

TABLE XIV. INTERPRETATION OF VALUES FOR IC

| Value | Interpretation |
|---|---|
| Less than 1 | Within the service design not all possible information is available |
| 1 | All possible information is available |

For example, the information content, thus the metric IC for the service interface depicted in Figure 10 is 0.8. The value of IC can be increased and maximized by adding the interaction protocol to the service interface.

## C. Loose Coupling

The loose coupling focuses on the reduction of dependencies between services within a service-oriented architecture and represents one of the most widespread aspects. A loose coupling promotes the scalability, fault tolerance, flexibility, and maintainability of the architecture [6, 7, 8, 16, 20]. Once a service requires another service to fulfill its functionality, a certain kind of coupling exists. However, in order to decrease the coupling, the following aspects can be considered that represent quality indicators for a loose coupling.

*1) Asynchronity:* According to Josuttis [7] and Maier et al. [11], long-running operations should be performed asynchronously. This means that the service consumer is being informed when the service provider has performed the called operation. This decouples the service consumer from the service provider during the execution of the operation.

In SoaML the communication mode is determined during the specification phase, i.e. this quality indicator can be evaluated on the basis of service designs. Within the interaction protocol, the IT architect can decide whether to provide an operation synchronously or asynchronously. For this purpose the attribute “IsSychronous” of the CallOperationActions within the Activity that represents the interaction protocol can either be set true or false. Thus, to determine this quality indicator the rate of long-running operations that are also asynchronous has to be measured.

$$ASYNC(s) = \frac{\left|ASO\left(IP(SI(s))\right) \cap LRO\left(O\left(RI(SI(s))\right)\right)\right|}{\left|LRO\left(O\left(RI(SI(s))\right)\right)\right|}$$

TABLE XV. VARIABLES AND FUNCTIONS USED FOR ASYNC

| Element | Description |
|---|---|
| ASYNC | Asynchrony |
| ASO(ip) | Asynchronous Operations: asynchronous operations within the interaction protocol ip |
| LRO(o) | Long Running Operations: long-running operations out of the set of operations o |

The metric is only valid if there is at least one long-running operation. The following table shows the interpretation of the values based on the ordinal scale. The metric returns values from 0 to 1.

TABLE XVI. INTERPRETATION OF VALUES FOR ASYNC

| Value | Interpretation |
|---|---|
| Less than 1 | There are long-running operations that are not provided asynchronously |
| 1 | All long-running operations are provided asynchronously |

The following figure shows the interaction protocol for the service interface depicted in Figure 10. Assumed that the Update operation is a long-running operation, ASYNC returns 0 as Update is not provided asynchronously.

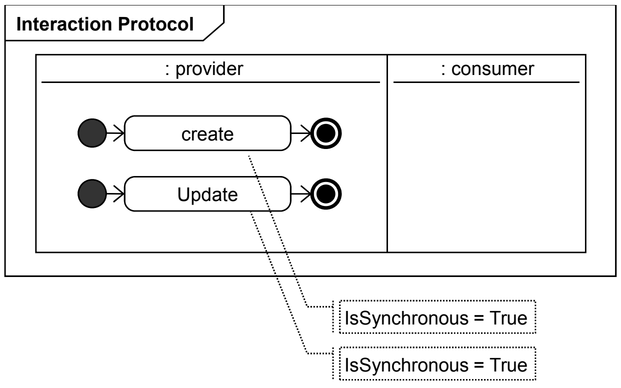

Figure 11. Example for asynchronity

*2) Common Data Types Complexity:* The usage of common data types across several services increases the coupling between them. If a service provider aspires to change the data types used by provided services, all other services that require this service have to be adapted too. Thus, Josuttis [7] advises to only use common data types if they are simple data types, such as String or Integer. Otherwise, the services should use own data types and the infrastructure should handle the transformation.

For this purpose, in SoaML the service designs have to be considered. The used parameters within the operations of a certain service and the contained data types should not be identical to the data types used by other services. A typical way to avoid this issue is to create identical data types, however within different UML packages for each created service. The following metric measures the ratio of common and simple data types to all used data types. For an optimal value, either there is no common data type or all commonly used data types are simple. The metric is only valid if the required artifacts, such as operations and data types, exist within the service design.

$$CDTC(s) = \frac{\left| SDT\left( \begin{array}{c} DT\Big(P\Big(O\big(RI(SI(s))\big)\Big)\Big) \cap \\ DT\Big(P\Big(O\big(RI(SI(ALL_s \setminus s))\big)\Big)\Big) \end{array} \right) \right|}{\left| DT\Big(P\Big(O\big(RI(SI(s))\big)\Big)\Big) \right|}$$

TABLE XVII. VARIABLES AND FUNCTIONS USED FOR CDTC

| Element | Description |
|---|---|
| CDTC | Common Data Types Complexity |
| SDT(p) | Simple Data Types: simple data types within the parameters pt |

The metric CDTC returns values from 0 to 1. The interpretation based on the ordinal scale is shown in the following table.

TABLE XVIII. INTERPRETATION OF VALUES FOR CDTC

| Value | Interpretation |
|---|---|
| 0 | There are no common data types used |
| Between 0 and 1 | There are common and complex data types used |
| 1 | The commonly used data types are simple |

The optimal value is represented by 0 or 1. A value between 0 and 1 should be avoided.

*3) Abstraction:* To decrease the coupling between service consumer and service provider, a service consumer should be able to use provided functionality without knowledge about the internal behavior of the service provider. This enables the invocation of functionality without knowledge about the implementation and an easier replacement of the implementation or the service provider. According to Erl [3], Josuttis [7] and Maier et al. [11], the operations should be designed in an abstract manner and should hide internal details. Thus, the quality indicator can be broken down into two quality indicators: First, the operations provided by the service should be abstract, i.e. the name and their purpose should be abstract. Additionally, the used parameter should be abstract, i.e. implementation details should not be exchanged when invoking an operation.

Thus, in SoaML the first quality indicator focuses on the operation of the interface that is realized by the service interface. The abstract operations are related to all provided operations. The second quality indicator regards the parameters that are used within the operations. The metric measures the ratio of abstract parameters to all parameters.

$$AO(s) = \frac{\left| A\Big(O\big(RI(SI(s))\big)\Big) \right|}{\left| O\big(RI(SI(s))\big) \right|}$$

$$AP(s) = \frac{\left| A\Big(P\Big(O\big(RI(SI(s))\big)\Big)\Big) \right|}{\left| P\Big(O\big(RI(SI(s))\big)\Big) \right|}$$

TABLE XIX. VARIABLES AND FUNCTIONS USED FOR AO AND AP

| Element | Description |
|---|---|
| AO | Abstraction of Operations |
| AP | Abstraction of Parameters |
| A(o) | Abstract: set out of operations o that are abstract |
| A(p) | Abstract: set out of parameters p that are abstract |

For AO and AP values from 0 to 1 are returned that can be interpreted on the basis of the ordinal scale as follows.

TABLE XX. INTERPRETATION OF VALUES FOR AO AND AP

| Value | Interpretation |
|---|---|
| Less than 0 | There exist operations respectively parameters that are not abstract |
| 1 | All operations respectively parameters are abstract |

According to the table above, a value of 1 is desired for both metrics because this represents that all operations respectively parameters are abstract which promotes the loose coupling. The following figure shows an interface that is realized by a service interface. One of the provided operations is not abstract, thus the value for AO is 0.5. The value of AP is 0.4 for five parameters are specified and only two of them are abstract.

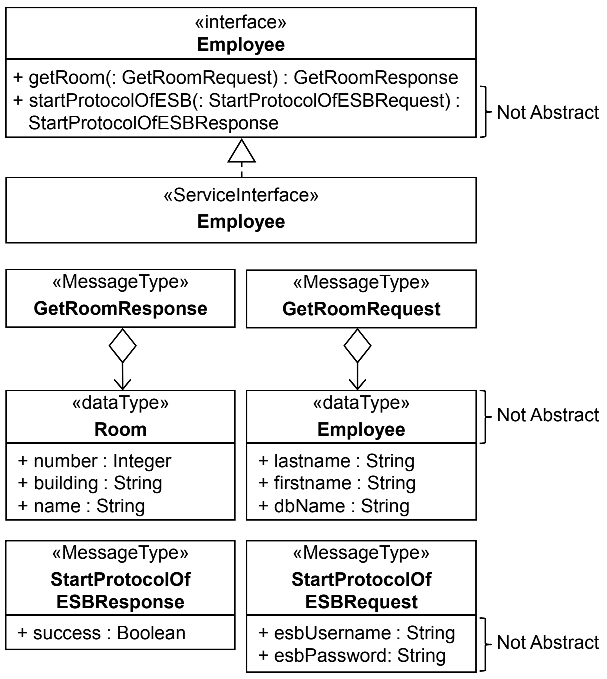

Figure 12. Example for abstraction

*4) Compensation:* According to Josuttis [7] and Engels et al. [8], in order to undo operations of a service that perform state-changing functionality, appropriate compensation operations should be provided. This enables the application of this service within transaction contexts, which requires an undo even if other services are the reason for a failure.

This quality indicator can already be measured on the basis of service candidates. For every operation candidate that represents a state-changing operation, an appropriate compensating operation candidate should exist. The metric first determines operation candidates that are not mainly compensating and change a state. Afterwards, out of this set the operation candidates are identified for those a compensating operation candidate exists. This set is related to the set of all non-compensating and state-changing operation candidates.

$$CF(sc) = \frac{\left|CFP\left(SC\left(NC\left(OC(sc)\right)\right)\right)\right|}{\left|SC\left(NC\left(OC(sc)\right)\right)\right|}$$

On the basis of service designs, instead of operation candidates the operations within the realized interface are considered.

$$CF(s) = \frac{\left|CFP\left(SC\left(NC\left(O\left(RI\left(SI(s)\right)\right)\right)\right)\right)\right|}{\left|SC\left(NC\left(O\left(RI\left(SI(s)\right)\right)\right)\right)\right|}$$

TABLE XXI. VARIABLES AND FUNCTIONS USED FOR CF

| Element | Description |
|---|---|
| CF | Compensating Functionality |
| NC(oc) | Non-Compensating: non-compensating operation candidates out of the set of operation candidates oc |
| NC(o) | Non-Compensating: non-compensating operations out of the set of operations o |
| SC(oc) | State Changing: operation candidates out of the set of operation candidates oc that provide state-changing functionality |
| SC(o) | State Changing: operations out of the set of operations o that provide a state-changing functionality |
| CFP(oc) | Compensating Functionality Provided: operation candidates out of the set of operation candidates oc a compensating operation candidate exists for |
| CFP(o) | Compensating Functionality Provided: operations out of the set of operations o a compensating operation exists for |

The metric CF returns values from 0 to 1. The interpretation on the basis of the ordinal scale is shown below.

TABLE XXII. INTERPRETATION OF VALUES FOR CF

| Value | Interpretation |
|---|---|
| Less than 0 | There exist state-changing operation candidates respectively operations without compensating operations candidates respectively operations |
| 1 | For all operation candidates respectively operations that provide state-changing functionality a compensating operation candidate respectively operation exists |

For the service interface depicted in Figure 10 the value of CF is 0.5 because for Update there exists a compensating operation, the Update operation itself. However, for the create operation there is no compensating operation. In order to increase the value of CF and thus the loose coupling, a delete operation should be added that enables the deletion of a prior created room.

### D. Autonomy

The autonomy of a service addresses its independence from other services [3, 7]. For increasing the autonomy of a service dependencies to other services have to be reduced. For the autonomy the following quality indicators can be identified.

*1) Service Dependency:* According to Erl [3], the direct dependencies to other services should be decreased. If a service depends from other services also its reliability, performance and predictability is influenced by these services.

In SoaML the direct dependencies can be evaluated on the basis of the usage dependencies between service candidates and the RequestPoints within service components. Both represent the dependencies of a service to other services in order to fulfill its functionality.

$$SD(sc) = |\, RS(sc)|$$

$$SD(s) = \left|\, RS\big(SCT(s)\big)\, \right|$$

TABLE XXIII. VARIABLES AND FUNCTIONS USED FOR SD

| Element | Description |
|---|---|
| SD | Service Dependency |
| RS(sc) | Required Services: service candidates the service candidate sc depends on |
| SCT(s) | Service Component: service component of the service s |
| RS(sct) | Required Services: services the service component sct depends on |

SD returns values from 0 to unlimited. The interpretation is based on the absolute scale.

TABLE XXIV. INTERPRETATION OF VALUES FOR SD

| Value | Interpretation |
|---|---|
| 0 | the service candidate or the functionality fulfilling service component depends on no other service candidate respectively service |
| n (n > 0) | the service candidate or service component requires n other services to fulfill its functionality |

For a maximal autonomy the value of SD should be 0. However, especially in the context of service-oriented architectures the reuse of existing functionality should be considered, which decreases the autonomy. Additionally, for improving other quality attributes, such as the unique categorization, the autonomy often has to be reduced too. The following figure shows a service component that provides the service CampusGuide. Due to the five RequestPoints, the metric SD for CampusGuide returns 5.0.

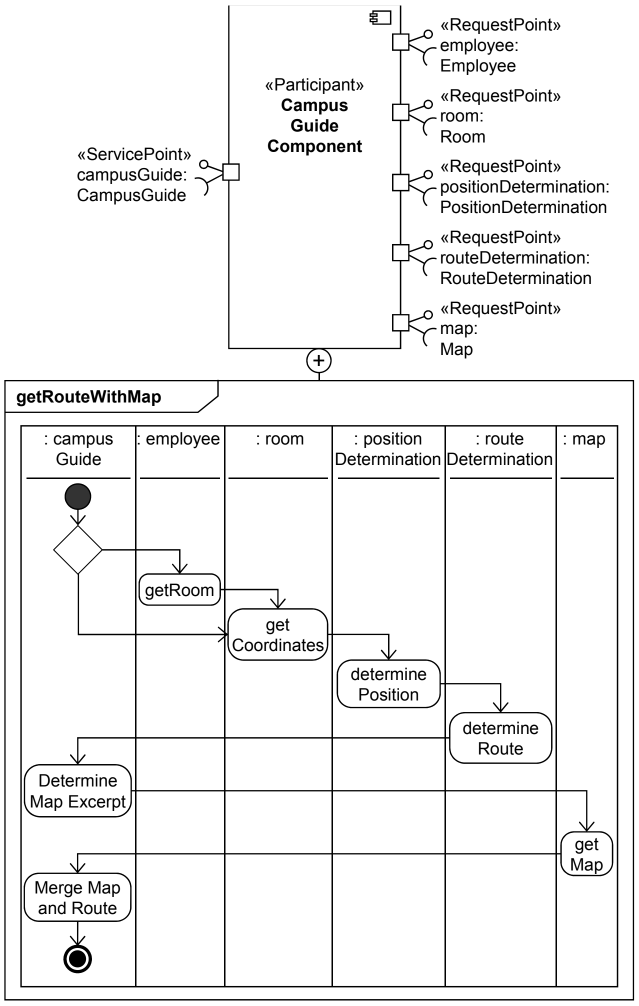
«ServicePoint» campusGuide: CampusGuide
«Participant» Campus Guide Component
«RequestPoint» employee: Employee
«RequestPoint» room: Room
«RequestPoint» positionDetermination: PositionDetermination
«RequestPoint» routeDetermination: RouteDetermination
«RequestPoint» map: Map
getRouteWithMap
: campus Guide
: employee
: room
: position Determination
: route Determination
: map
getRoom
get Coordinates
determine Position
determine Route
Determine Map Excerpt
get Map
Merge Map and Route

Figure 13. Example for service dependency

*2) Functionality Overlap:* According to Erl [3], a clear specification of the functional boundary of a service increases its autonomy. This means that the functionality of a service should not overlap with functionality of other services. Background of this requirement is that an overlap often results in dependencies between these services. In order to use functionality of a certain service, due to the overlap, also the functionality of the other services is necessary. Thus, the service with overlapping functionality can only be used together with other services. For avoiding functionality overlaps, services should be normalized [4].

The functionality overlap can be determined both on service candidates and service designs. For evaluating a service candidate its operation candidates have to be compared with the operation candidates of other service candidates. Afterwards, the set of operation candidates that provide redundant functionality are related to all provided operation candidates.

$$FO(sc) = \frac{|\, RF\big(OC(sc), OC(ALL_{sc} \setminus sc)\big)\, |}{|\, OC(sc)\, |}$$

On the basis of service designs, the functionality overlap is determined by means of the operations within the realized interface.

$$FO(s) = \frac{\left| RF\left(O\left(RI(SI(s))\right), O\left(RI(SI(ALL_s \setminus s))\right)\right)\right|}{\left| O\left(RI(SI(s))\right)\right|}$$

TABLE XXV. VARIABLES AND FUNCTIONS USED FOR FO

| Element | Description |
|---|---|
| FO | Functionality Overlap |
| RF(oc1, oc2) | Redundant Functionality: operation candidates out of the set of operation candidates oc1 with redundant functionality to the operation candidates oc2 |
| RF(o1, o2) | Redundant Functionality: operations out of the set of operations o1 with redundant functionality to the operations o2 |

The metrics return values from 0 to 1. The following table shows the interpretation of the values based on the ordinal scale.

TABLE XXVI. INTERPRETATION OF VALUES FOR FO

| Value | Interpretation |
|---|---|
| 0 | The operation candidates respectively operations of the considered service candidate or service do not provide functionality that overlaps with functionality of other service candidates or services |
| Between 0 and 1 | The operation candidates respectively operations of the considered service candidate or service provide functionality that overlaps with functionality of other service candidates or services |
| 1 | The operation candidates respectively operations of the considered service candidate or service provide only functionality that overlaps with functionality of other service candidates or services |

Based on the service candidates depicted in Figure 9, the following figure shows a service candidate with functionality overlap. The metric returns 0.5, for half of the provided operation candidates overlap with functionality provided by the Facility Management service candidate.

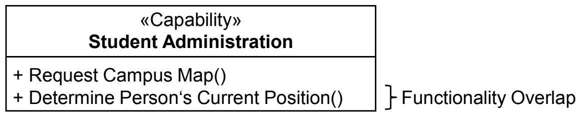

Figure 14. Example for functionality overlap

## IV. CONCLUSION AND OUTLOOK

In this article we presented metrics for evaluating service designs based on SoaML. The approach uses the textual descriptions for quality attributes as introduced in existing work and analyses these quality attributes with regard to quality indicators, which are measureable on service candidates as preliminary services and fully specified service designs. The concept of a service design was derived from existing development processes. For each of the quality indicators formalizations were given that reuse the concepts of service candidates and service designs and their specification in SoaML. This enables the application of the formalization and thus the determination of the quality indicators on modeled service designs without additional interpretation effort.

The identification of quality indicators and their formalizations help IT architects to comprehensibly evaluate service designs with regard to common and widespread quality attributes. In this article the quality attributes of a unique categorization, loose coupling, discoverability, and autonomy were considered. Other quality attributes and potential quality indicators were informally introduced in [1]. The usage of SoaML as language to model service candidates and service designs enables the integration of the metrics into existing development tools. SoaML represents an emerging standard for modeling service-oriented architectures. Its availability as XMI [45] enables the usage in any UML-capable development tool.

To illustrate the metrics, service candidates and service designs of a service-oriented campus guide system as it is developed at the Karlsruhe Institute of Technology (KIT), called KITCampusGuide, have been introduced. Well-chosen excerpts of service candidates and service designs for this scenario were used to apply the metrics and thus to show their validity. The metrics were applied in practice to design services for the KITCampusGuide with a unique categorization, loose coupling, discoverability, and autonomy. Currently, the metrics also applied for the domain campus management in order to create a catalog of services for universities and their administrative processes according to the Bologna Process [36] with same quality attributes. In this context especially the compliance with naming conventions and the usage of a common domain model for a functional naming are of interest. Additionally, the metrics are applied at the Personalized Environmental Service Configuration and Delivery Orchestration (PESCaDO) project [40, 41], a project co-funded by the European Commission. Also in this case, service designs are supposed to be created that verifiably fulfill the four introduced quality attributes.

Additionally to the identification and formalization of metrics, we work on integrating the metrics into development tools in order to further support the IT architect during the design phase. A more detailed formalization based on the Object Constraint Language (OCL) [44], as already demonstrated in [1], enables the embedding of well-chosen metrics into UML tools and thus an automatic evaluation of service designs. For this purpose, additional semantic information may be necessary. Hence, we are also working on a determination and formalization of this additional information. Furthermore, we work on an integration of the metrics into existing development processes. The metrics are supposed to support the IT architect in creating service designs with certain quality attributes by identifying service designs flaws [2]. If service design flaws could be determined, appropriate action alternatives that may result in improved service designs are provided to the IT architect. These action alternatives help the IT architect to revise the service designs with regard to certain quality attributes.

## REFERENCES

[1] 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.

[2] 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.

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

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

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

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

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

[8] 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.

[9] B. Humm, O. Juwig, "Eine normalform für services", GI Informatik 2006, Dresden, Germany, October 2006, pp. 99-110.

[10] S. Cohen, "Ontology and taxonomy of services in a service-oriented architecture", Microsoft Architecture Journal, 2007.

[11] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, T. Winterberg, „Lose kopplung – warum das loslassen verbindet", SOA-Spezial, Software & Support Verlag, 2009.

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

[13] B. Maier, H. Normann, B. Trops, C. Utschig-Utschig, 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 H. Schmidt, "Cohesion metrics for predicting maintainability of service-oriented software", Seventh International Conference on Quality Software (QSIC 2007), 2007.

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

[17] D. Rud, S. Mencke, A. Schmietendorf, R. R. Dumke, „Granularitätsmetriken für serviceorientierte architekturen, MetriKon, 2007.

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

[19] S. W. Choi, 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.

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

[21] 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]

[22] 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.

[23] 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]

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

[25] 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.

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

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

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

[29] B. Boehm, Characteristics of Software Quality, Elsevier Science Ltd, 1978. ISBN 978-0444851055.

[30] J. McCall, P. Richards, and G. Walters, "Factors in software quality – volume 1", 1977.

[31] ISO/IEC, "ISO/IEC 9126-1:2001 software engineering: product quality – quality model", 2001.

[32] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software-engineering measurement", IEEE Transactions on Software Engineering, Vol. 22, No. 1, 1996.

[33] 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]

[34] 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]

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

[36] 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]

[37] 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.

[38] 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.

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

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

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

[42] 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]

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

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

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