Evaluation of Service Designs Based on SoaML

Michael Gebhart, Marc Baumgartner, Stephan Oehlert, Martin Blersch, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{ gebhart | baumgartner | oehlert | blersch | abeck } @kit.edu

Abstract— **In the context of service-oriented architectures, services are expected to fulfill certain service characteristics, such as high autonomy or loose coupling. In order to easily influence the design of these services, it is desirable to evaluate their characteristics early on in the development process, i.e. during design time. Related work focuses on the description of desired service characteristics that refer to services as a whole and does not address the evaluation of service designs in terms of their characteristics. Thus, in this paper, we analyze common and widespread service characteristics, derive evaluable design attributes that refer to elements of service designs based on SoaML, and demonstrate the formalization of an exemplarily design attribute using OCL. The application of the identified design attributes on a tentative service design of a service-oriented surveillance system helps to create a revised service design with improved service characteristics.**

Keywords-service design; soaml; evaluation; service characteristic; design attribute

I. INTRODUCTION

Today, several organizations are shifting their information technology (IT) to service-oriented architectures. In this context, services provide the functionality that is required to support the business of the organization. With the shift to service-oriented architectures, goals concerning the IT, such as increased flexibility and better alignment with the business, are often associated [1] in order to quickly react to changing business requirements. To attain these goals, service characteristics have been identified that services should fulfill. Such characteristics include high autonomy or loose coupling [1, 4].

Since changing the design of services after their implementation and deployment is costly and complicated, it is preferable to analyze services regarding the common and desired service characteristics during design time. Each service characteristic can be divided into a pair consisting of a service attribute and its value. For example, the service characteristic "loose coupling" is composed of the service attribute "coupling" and its value "loose". To evaluate a service attribute that refers to the service as a whole, it has to be broken down into a set of evaluable design attributes that refer to elements of a service design, such as the provided service interface [3] or the service component [23] that realizes the functionality and contains the service logic.

Existing work by Erl [1], Reussner et al. [4], Josuttis [5], Engels et al. [6], and Maier et al. [7, 8, 9] focuses on the description of desired service characteristics. It introduces a comprehensive set of service characteristics that services within a service-oriented architecture should follow. However, it is not obvious how a characteristic such as loose coupling is reflected in the design of a service, and the authors do not specifically address how to evaluate the fulfillment of service characteristics. Furthermore, their work does not explicitly describe design attributes that refer to elements of a service design or metrics. Other work emphasizing metrics in the context of service-oriented architecture, as introduced by Perepletchikov et al. [10, 12], Rud et al. [13], Hirzalla et al. [14], and Choi et al. [15], is only partly applicable for evaluating service designs with respect to service characteristics because some metrics require more information than is actually available during the design phase. In other cases, the relation of the measured design attribute to the desired service characteristics is not apparent, which reduces the motivation to apply this metric. Additionally, existing metrics are most often described in merely conceptual terms and are not applied on a commonly used service design model. This hampers the usage of these metrics because the concepts within the metrics, such as "number of clients", first have to be correctly interpreted and then mapped onto representations of themselves based on the service design model.

The contribution of this paper is the direct derivation of evaluable design attributes from common and widespread service characteristics. The derived attributes refer to certain elements within a service design modeled with the Service-oriented architecture Modeling Language (SoaML) [3] in order to evaluate service designs during design time. SoaML was chosen as the language for the service design because it is a standardized UML profile [28] and metamodel for describing and formalizing service-oriented architectures and because it is becoming increasingly accepted and employed. To determine evaluable design attributes, we first introduce the notion of a service design itself and define its elements in SoaML. In a next step, we analyze a comprehensive set of service characteristics and derive design attributes that refer to elements of a service design in SoaML. Since some of the design attributes can be quantified and automatically measured, while others require intuition, we demonstrate a formalization with a design attribute that affects the autonomy of a service. The design attribute is formalized using the Object Constraint Language (OCL) [29]. This enables the automatic measurement of the design attribute on a service design based on SoaML.

A subset of the identified design attributes is illustrated by a service design of a service-oriented system for a network-enabled surveillance and tracking developed at the

Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, called N.E.S.T [27]. Here, services are developed for data processing and information analysis that can be combined to high level services for automating tasks, such as a detecting abnormal human behavior or tracking suspicious persons. We evaluate a tentative service design within this scenario using the identified design attributes and demonstrate how this evaluation can be used to create an improved service design.

The paper is organized as follows: Section 2 presents the related work in the context of service design formalization, service characteristics, and metrics applicable within service-oriented architectures. In Section 3, the notion of a service design is introduced based on SoaML. Afterwards, evaluable design attributes are derived from common and widespread service characteristics. A subset of these attributes is applied on a tentative service design of N.E.S.T. and the usage of this evaluation to create an improved service design is shown. Finally, one identified design attribute influencing the autonomy of a service is formalized as an executable OCL statement. Section 4 concludes the paper and offers suggestions for future research.

II. Related Work

In Erl [1, 2], a service design is introduced as the result of the service design process that is performed for every service. There are different design processes for entity, task, and utility services extended by the best practices introduced in [21, 22]. Entity, task, and utility services differ in their functional scope. Each process contains various steps in which the service and its logic are designed and states which information about a service should be expected as the result of executing these steps. The resulting information that is available about a service is summarized as a service design. It includes a description of the provided service interface, the internal service logic, and the potentially required services. The design processes use established standards, such as WSDL [32], to describe the services. We use the processes as a guideline for building services and see the notion of service design as useful. However, formal models that platform independently specify this information are missing. We introduce the formalization of a service design as a service design model. The term "service design model" is derived from the term "service model" [18, 19] as introduced in the Service-Oriented Modeling and Architecture (SOMA) [16, 17] as an extension of the Rational Unified Process (RUP) [20] and the term "service design" as described in Erl.

Service Component Architecture (SCA) [23] similarly describes the creation of service-oriented solutions as the composition of services. Functionality is provided by an interface and the service is implemented by means of a service component that requires other services. This conforms to the service design as introduced in Erl [1, 2]. The term "service component" as a realization of a service is also introduced in RUP SOMA [18, 19]. Hence, we reuse the notion of a service component as the realizing component of a service.

The Service-oriented architecture Modeling Language (SoaML) [3] is an emerging standard from the OMG for a UML profile and metamodel for modeling service-oriented architectures, focusing on services and how they relate to one another. SoaML is heavily based on the UML composite structure metamodel [28]. The standard describes the content of service designs, the provided ServiceInterface element, a Participant element containing the service logic, and the required ServiceInterface elements similar to Erl [1]. Even though SoaML is currently only available as a preliminary beta version, due to its increasing acceptance and employment, we chose SoaML to formalize service designs. However, SoaML focuses on the description of modeling elements and does not explain how to evaluate services.

According to Erl [1], Reussner et al. [4], Josuttis [5], Engels et al. [6], and Maier et al. [7, 8, 9], a service should fulfill service characteristics, such as a well-defined service interface, loose coupling, or high autonomy. They list the desired service characteristics and provide comprehensive textual descriptions. However, they do not explain how to exactly evaluate a service design in terms of their characteristics or provide a formal description of the characteristics and their impact on the concrete elements of a service design. We see these characteristics as valid and reuse their descriptions to derive design attributes that refer to concrete elements of a service design and can be evaluated during the design phase.

Metrics are a widely used approach to assess software quality based on measuring the artifacts that result from the development process. There are a number of works proposing metrics for measuring the attributes of service-oriented software. Perepletchikov et al. [11] extend the generic software model of Briand [30] to propose a formal model for "structural and behavioral properties" of service-oriented software and introduce metrics for measuring cohesion [10] and coupling [12]. Rud et al. [13] describe metrics for measuring the granularity of services; Hirzalla et al. [14] focus on flexibility. Choi et al. [15] describe metrics for the reusability of services. However, their work is only partially applicable for evaluating service designs because the metrics discussed are mostly meant for application on an entire service-oriented architecture with fully implemented services, i.e. they require more information than is available within service designs. Additionally, some metrics are not related to common and widespread service characteristics, which hampers the incentive to measure them. Since no common modeling language is used, their definitions require interpretation about how to use them with common service design models, as for instance SoaML. Therefore, we reuse the work cited above as essential input on how to evaluate services, though we base our specifications of design attributes directly on a concrete modeling language, namely SoaML, and derive them from common and widespread service characteristics.

Software metrics often measure source code as the primary development artifact, thus inhibiting their application on models. Due to the proliferation of models based on metamodels as first class development artifacts, as introduced in model-driven development approaches, such as MDA [31], new techniques have been proposed to measure models directly. Reynoso et al. [24] show how

metamodeling techniques can be used to formalize metrics on models as metamodel instances using OCL [29]. Monperrus et al. [26] describe a modeling approach for metrics based on the custom metamodel called MDM. In [25] they describe a generic metric definition approach called sigma. Model metrics are then defined as specializations of this sigma metric and can be applied with filtering functions on custom models, which enables decoupling metric definitions from these said models. While this enables the formalization of more generalized metrics, it also introduces an additional step of indirection which we prefer to avoid. We thus reuse the concept of formalizing metrics with OCL because OCL is an established and sound language for querying UML models.

III. Evaluation of Service Designs Based on SoaML

This section introduces service designs based on SoaML and derives evaluable design attributes from common and widespread service characteristics. After we introduce the notion of a service design illustrated by a service of N.E.S.T. in Section A, in Section B we exemplify the derivation approach with the autonomy service attribute. Section C summarizes all identified design attributes and applies a subset to the previously introduced service design of N.E.S.T. In Section D, the results of the evaluation are used to revise the service design of the prior sections in order to create an improved service design. Section E demonstrates the automatic measurement of a design attribute that affects the autonomy of a service using OCL.

A. Service Designs Based on SoaML

According to Erl [1, 2], a service design consists of a provided service interface, the internal service logic, and potentially required services. In SCA [23] and RUP SOMA [18, 19], the service logic is implemented by the service component that realizes a service. Combined, a service design consists of a provided service interface, the implementing service component, and the required services. The provided service interface is externally visible to service consumers. The service component and potentially required service interfaces are part of the internal view and are thus not visible for service consumers. However, this information impacts the service characteristics, and is therefore an important part of a service design.

In SoaML, an element ServiceInterface exists that correlates with our understanding of a service interface. It is defined as the type of a ServicePoint or RequestPoint. As the type of a ServicePoint, ServiceInterfaces represent provided service interfaces and as the type of a RequestPoint they represent required service interfaces, i.e. required services. The service component is represented as a Participant in SoaML. Thus, the concepts of a service design can be directly mapped to SoaML. Figure 1 shows a modeled service design of a tentative draft of the TaskExecuter service in N.E.S.T. using SoaML as UML profile on a high level view.

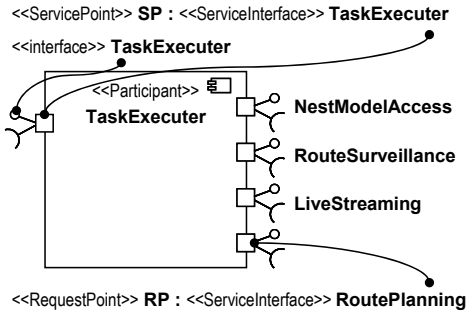

Figure 1. Tentative TaskExecuter service design in SoaML

The TaskExecuter service in N.E.S.T. enables surveillance and tracking of a person by providing the service interface TaskExecuter as the type of the ServicePoint SP. The provided technical interface TaskExecuter describes the provided operations.

The service component is realized as a Participant TaskExecuter. It includes the internal service logic and requires four services. For person surveillance, the allowed routes have to be planned (RoutePlanning), video has to be streamed (LiveStreaming), the planned route has to be surveilled (RouteSurveillance), and a virtual model of the person to be tracked and his or her current position has to be accessed (NestModelAccess). The final tracking and surveillance of the person is part of the TaskExecuter's internal logic. Each of the required services is described as a ServiceInterface as the type of a RequestPoint.

A ServiceInterface, both provided and required, can comprise a technical interface that the service provides and a required technical interface that a service consumer has to provide in order to receive callbacks. A technical interface is a collection of signatures of operations. The signature contains the name of the operation, the parameters and their respective names and parameter types, and the return type of the operation. The types used here can be either primitive (i.e. atomic) or, as preferred in the context of service-oriented architecture, complex message types. Additionally, a ServiceInterface can and should describe the capabilities the service exposes as well as the allowed interactions between the service provider and service consumer, which are called an interaction protocol. Figure 2 shows the tentative ServiceInterface for TaskExecuter in SoaML. In this case, the ServiceInterface only provides information about the provided technical interface and the capabilities.

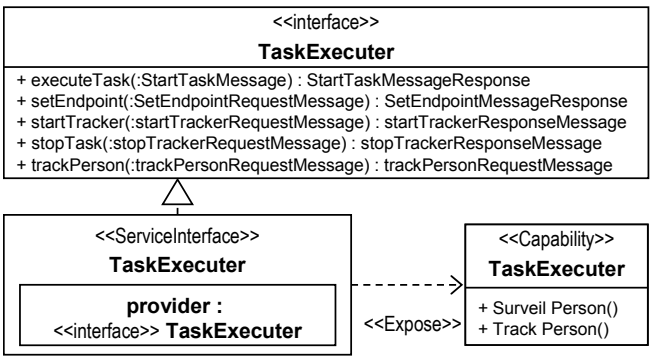

Figure 2. Tentative TaskExecuter service interface in SoaML

B. Derivation of Evaluable Autonomy Design Attributes

After creating a service design as defined for the tentative TaskExecuter service in Figures 1 and Figure 2, it is desirable to evaluate the service design regarding certain service characteristics, such as a well-defined service interface, loose coupling, or high autonomy. This enables the identification of design flaws whose revision may result in an improved service design. For this purpose, it is necessary to break the correlating service attributes down into design attributes that refer to elements of a service design. This means that it has to be determined if all required information is available during design time and if it is part of a service design based on SoaML. To demonstrate the approach of how to determine the design attributes, in the following the tentative TaskExecuter service is evaluated with respect to its autonomy.

According to Erl [1], a service is highly autonomous if the following criteria are fulfilled: The functional boundary should not overlap with other services, services are deployed in an environment over which they exercise a great deal of control, service instances are hosted by an environment that accommodates high concurrency for scalability purposes, and the number of required services should be minimal.

Now, each criterion is analyzed stepwise. The first criterion, i.e. that the functional boundary should not overlap with other services, means that at design time and transferred to SoaML the capabilities of a given ServiceInterface should not overlap with the capabilities of another ServiceInterface. To illustrate this aspect, Figure 3 shows all other ServiceInterface elements of N.E.S.T. besides TaskExecuter and their capabilities. There is an overlap of the capabilities exposed by the TaskExecuter ServiceInterface with the capabilities of the PersonTracking ServiceInterface: The capability "trackPerson" is exposed by both ServiceInterfaces. Thus, the TaskExecuter does not optimally fulfill this criterion. This shows that this criterion is evaluable during design time and refers only to information available within a service design. Thus, this criterion is appropriate as a design attribute.

The second criterion, which concerns the deployment in an environment over which they exercise a great deal of control, is not evaluable during design time and is not part of a service design. Thus, this criterion is not suitable as a design attribute.

The third criterion for high autonomy – service instances are hosted by an environment that accommodates high concurrency for scalability purposes – is also not evaluable during design time and not part of a service design. Thus, this criterion is not considered as a design attribute either.

The fourth and last aspect covers the dependencies of the service to other services. This can be evaluated at design time by counting the number of RequestPoints. The more services are required, the less the autonomy is. Figure 1 shows the service component implementing the Task Executer Service and its required service interfaces by means of RequestPoints. Since it requires four services, the Task Executer service is not maximally autonomous regarding this design attribute.

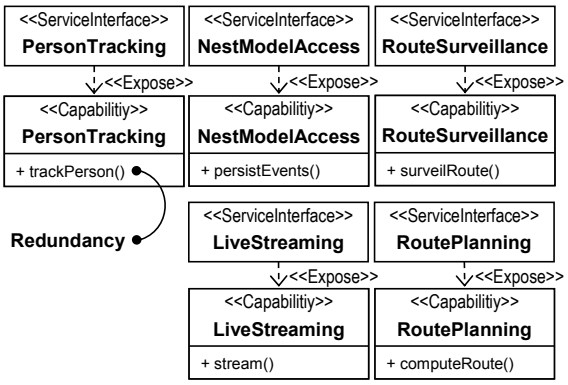

Figure 3. Capabilities of other services in N.E.S.T.

C. Summary of Evaluable Design Attributes

The approach for deriving design attributes can be applied on all service characteristics as identified in Erl [1], Reussner et al. [4], Josuttis [5], Engels et al. [6], Maier et al. [7, 8, 9], and SoaML [3]. The design attributes are summarized in the following table. For each attribute, the source from which it was derived is given.

TABLE I. SUMMARY OF IDENTIFIED DESIGN ATTRIBUTES

Design Attribute	***Preferred Characteristic in SoaML***
Autonomy	
Capability Redundancy [1]	There is no capability within Capability elements of a ServiceInterface that is redundant to capabilities of any other ServiceInterface.
Depending Services [1]	There is no RequestPoint at the Participant.
Service Interface Design	
Service Interface Extent [1]	All possible information (capabilities, interaction protocol, required / provided technical interface) is given by the provided ServiceInterface.
Service Interface Formalization [1, 4]	A ServiceInterface element exists.
Data Model Consistency [1]	The operations within the provided technical interface use data types of a common data model.
Convention Compliance [1]	The ServiceInterface, its operations and parameters within the technical interfaces follow conventions, such as naming conventions.
Coupling	
Service Interface Asynchronity [5, 7]	A required technical interface exists and the interaction protocol describes preferred asynchronous interactions.
Service Interface Data Types [5]	The provided technical interface only uses simple data types instead of complex data types.
Service Interface Abstraction [1, 5, 7]	The provided technical interface only contains operations and parameters that hide implementation details.
Transaction Handling [1, 5, 7]	If the service logic of the Participant requires transactions, then the logic includes compensation functionality and / or the provided technical interface includes compensating operations.
Parameter Style [3]	The operations within the technical interfaces use message style parameters instead of Remote Procedure Call (RPC) style.
Abstraction	

Service Interface Abstraction [1, 5]	Equals "Service Interface Abstraction" in "Coupling".
Service Interface Formalization [1, 4]	Equals "Service Interface Formalization" in "Service Interface Design".
Reusability	
Service Component Agnosticity [1]	The logic of the Participant can be reused in several processes.
Service Interface Genericity [1]	The parameter of operations within the provided technical interface should be generic.
Concurrency [1]	The service logic of the Participant should enable a concurrent execution of the service.
Self-Containedness	
Capability Redundancy [1, 5]	Equals "Capability Redundancy" in "Autonomy".
Depending Services [1, 5]	Equals "Depending Services" in "Autonomy".
Operation Order [5, 6]	There are no dependencies (order) between operations within the interaction protocol of the provided ServiceInterface.
Statelessness	
Service Component State Management [1, 5]	The logic of the Participant does not include activities for saving the state within the Participant.
Operation Parameters [5]	The operations within the technical interfaces only contain parameter types of complete objects instead of IDs for objects.
Discoverability	
Convention Compliance [1]	Equals "Convention Compliance" in "Service Interface Design".
Functional Service Interface [1, 5, 7]	The provided technical interface only contains operations and parameters with functional context. These and the service itself are suitably named.
Composability	
Multiple Granularity [1]	There exist operations within the technical interfaces that allow similar functionality with different granularity.
Idempotency	
Multiple Operation Call Handling [1, 5, 8]	The logic of the Participant contains activities to handle multiple operation calls.
Classification	
Entity / Task Classification [1, 4, 6, 7, 8, 9]	All capabilities within the Capability element are either responsible for managing data of business entities (entity service) or keep business logic that only uses business entities (task service).
Service Interface Well-Definition	
Service Interface Extent [1]	Equals "Service Interface Extent" in "Service Interface Design".

To demonstrate the design attributes, a comprehensible subset based on the information provided in Figures 1 and 2 is applied on the tentative TaskExecuter service design. Since some information, such as the message details and the internal service logic in terms of activity diagrams, is hidden for the sake of simplicity, not all design attributes and their evaluations would be comprehensible.

TABLE II. EVALUATION OF TASKEXECUTER SERVICE

Design Attribute	*Applied on tentative TaskExecuter service*
Capability Redundancy	There is redundancy with the PersonTracking ServiceInterface.
Depending Services	There are several Request Points.
Service Interface Extent	The ServiceInterface does not provide an interaction protocol or an technical interface required by the service consumer.
Service Interface Formalization	A ServiceInterface element exists.
Convention Compliance	The parameters within the provided technical interface do not follow uniform conventions: "SetEndpointMessageResponse" compared to "stopTrackerRequestMessage".
Service Interface Asynchronity	Within the ServiceInterface, neither a required technical interface nor an interaction protocol exists. There is also no asynchronous interaction implemented.
Parameter Style	The operations only use message style parameters.
Service Component Agnosticity	The service logic is very process-specific.
Operation Order	There is no interaction protocol but there would be dependencies. For example, the setEndpoint operation has to be called after executeTask.
Functional Service Interface	The service provides functionality to surveil persons. However, it is named TaskExecuter.
Multiple Granularity	Within the provided technical interface for each functionality, there is only one operation.
Entity / Task Classification	There are only capabilities that keep complex business logic, thus the service is a task service.

D. *Revision of the Service Design*

Following the evaluation of a service design, the results can be used to create a revised version that better fulfills the desired service characteristics. In the following, a revised service design for the TaskExecuter service is created. To improve the autonomy, the redundant capability "trackPerson" is reused from the PersonTracking service even if more services are required. For improved discoverability and looser coupling, the service is now named for what it really does, person surveillance. Figure 4 shows the PersonSurveillance service in SoaML. To improve the service interface design, the service interface is extended to contain the interaction protocol and a required service interface description even if no callbacks are required. The messages are also convention compliant now. Figure 5 shows the new PersonSurveillance ServiceInterface.

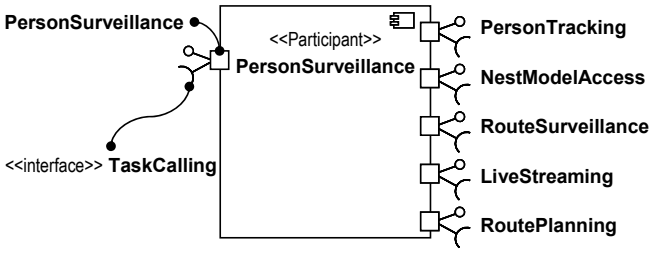

Figure 4. Revised TaskExecuter service in SoaML

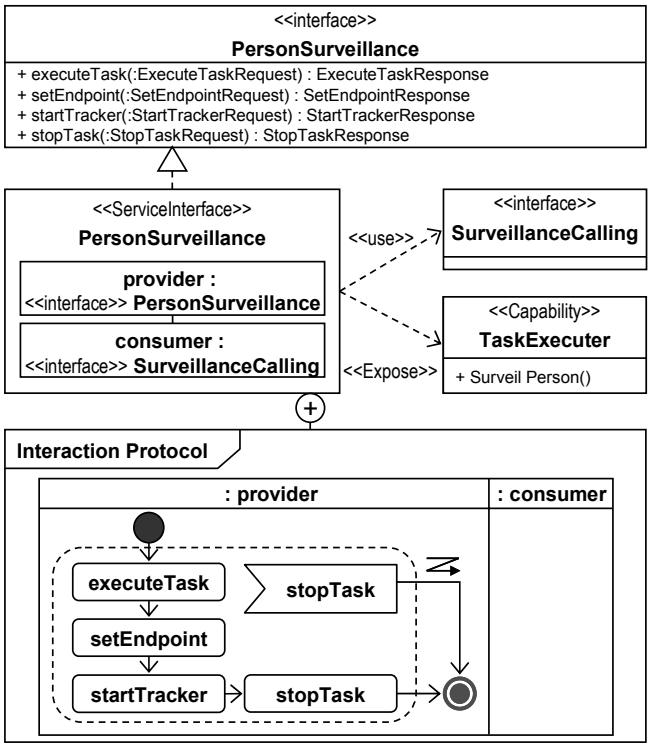

Figure 5. Revised TaskExecuter service interface in SoaML

E. Automatic Measurement Using OCL

While some of the identified design attributes require intuition, others, such as Capability Redundancy, can be quantified and directly measured. For this purpose, the design attribute can be transformed into an executable OCL statement. As proof of concept, the identified design attribute Capability Redundancy is formalized. The following measurement is used:

$$\frac{\textit{Number of redundant capabilities of the ServiceInterface}}{\textit{Number of all capabilities of the ServiceInterface}}$$

This returns the percentage of redundant capabilities. A value of 0 means that all capabilities are unique and 1 indicates that all capabilities overlap. As an executable statement in OCL the design attribute can be formalized as follows:

```
context ServiceInterface
let
getAllExposes : Set(Dependency) =
 Dependency.allInstances()->
  select(d|d.isStereotypeApplied
  (d.getApplicableStereotype('SoaML::Expose'))),
getAllCapabilityElements: Set(Class) =
 Class.allInstances()->select(c |
  c.isStereotypeApplied(c.getApplicableStereotype
  ('SoaML::Capability'))),
getOwnCapabilityElements: Set(Class) =
 getAllCapabilityElements->select(c |
  getAllExposes->exists(e | e.supplier->exists(s |
   s = c) and e.client->exists(cl | cl = self))),
getOtherCapabilityElements: Set(Class) =
 getAllCapabilityElements->select(c |
  getAllExposes->exists(e | e.supplier->exists(s |
   s = c) and e.client->exists(cl | cl <> self))),
getOwnCapabilities: Bag(String) =
 getOwnCapabilityElements->collect(c |
  c.ownedOperation->collect(o | o.name)),
getOtherCapabilities: Bag(String) =
 getOtherCapabilityElements->collect(c |
  c.ownedOperation->collect(o | o.name)),
getNumberOfOwnRedundantCapabilities: Integer =
 getOwnCapabilities->
  intersection(getOtherCapabilities)->size(),
getNumberOfOwnCapabilities: Integer =
 getOwnCapabilities->size()
in getNumberOfOwnRedundantCapabilities /
getNumberOfOwnCapabilities
```

When applied on the tentative service design in Figures 1 and 2, the OCL expression returns 0.5, but for the revised service design in Figures 4 and 5 it returns 0, the optimal value.

IV. Conclusion and Outlook

In this paper, we presented evaluable design attributes for service designs based on SoaML. Since a service design is one of the first design artifacts when developing a service, it strongly influences the service characteristics of the resulting service. Hence, we firstly defined the elements of a service design. To formalize a service design model, we chose the emerging standard SoaML from the OMG that represents a UML profile and metamodel for modeling service-oriented architectures. Afterwards, we analyzed common and widespread service characteristics and derived design attributes that can be evaluated already during design time on a service design model. Since some of them are quantifiable, it is partially possible to formalize executable OCL statements that automatically measure the design attributes on a SoaML-based service design. We demonstrated this formalization using one design attribute that affects the autonomy of a service.

The identified design attributes help IT architects to evaluate service designs during design time and thus to develop services more systematically with regard to their characteristics. Receiving early feedback about the expected service characteristics helps the IT architect to identify improvements prior to implementation, and thus easily work them in and exploit them. Additionally, several different service design alternatives can be quantified and compared. The service design can be iteratively evaluated and revised so that it better fulfills the desired service characteristics. This improvement in turn supports the attainment of goals concerning the IT that are associated with the establishment of a service-oriented architecture, such as increased flexibility or better alignment with the business.

Due to the usage of SoaML and OCL, common and standardized languages that are supported by widespread UML tools were applied in our paper. Though SoaML is a very new UML profile and metamodel and still under development, its employment and acceptance are increasing. Several tools already support SoaML. Additionally, SoaML reuses the UML profile mechanism, which is widely supported. The profile is already available as XMI [33],

enabling the identified design attributes to be applied in any UML-profile-capable development tool.

A comprehensible subset of the design attributes was exemplarily applied on a real-world service-oriented system for a network-enabled surveillance and tracking developed at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation. Instead of presuming service characteristics, we systematically used the design attributes to evaluate a tentative service design. This enabled us to identify potential design flaws and revise them in order to create a service design with improved characteristics.

In our future work, we plan to further utilize the identified design attributes within the entire development process of a service. On the one hand, this will include the use of the design attributes to support design decisions during the creation of a service design. On the other hand, we also plan to further improve the use of the design attributes as a tool to identify service design flaws. Our goal is to report the elements of a service design that should be revised to the IT architect and to list the design decisions that should be reconsidered to improve the service design. Additionally, we will work on formalizing further design attributes using OCL to enable tool support for the development of services. Ideally, development tools could automatically calculate the degree to which a service design possesses a given design attribute that affects a particular service characteristic and visually highlight the elements of a service design that should be revised to improve its characteristics. The entire approach will be applied to design services for the domain campus management, as required to integrate university systems, and for a human-centered environmental observation system developed at the Karlsruhe Institute of Technology.

REFERENCES

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

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

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

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

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

[6] 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.

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

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

[9] 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.

[10] 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.

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

[12] 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.

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

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

[15] 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.

[16] IBM, "RUP for service-oriented modeling and architecture", IBM Developer Works, http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/, 2006. [accessed: May 10, 2010]

[17] 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: May 10, 2010]

[18] S. Johnston, "UML 2.0 profile for software services", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/419_soa/, 2005. [accessed: May 10, 2010]

[19] S. Johnston, "Modeling web services, part 1", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/05/1129_johnston/, 2005. [accessed: May 10, 2010]

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

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

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

[23] Open SOA (OSOA), "Service component architecture (SCA), sca assembly model V1.00", http://osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf, 2009. [accessed: May 10, 2010]

[24] L. Reynoso, M. Genero, J. Cruz-Lemus, and M. Piattini, "Using ocl in the formal definition of ocl expression measures", Workshop on Quality in Modeling, Co-Located with MoDELS 2006, 2006.

[25] M. Monperrus, J.-M. Jézéquel, J. Champeau, and B. Hoeltzener, "Measuring models", 2008.

[26] M. Monperrus, J.-M. Jézéquel, J. Champeau, and B. Hoeltzener, "A model-driven measurement approach", MoDELS 2008, 2008.

[27] 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.

[28] OMG, "Unified modeling language, superstructure", Version 2.2, 2009.

[29] OMG, "Object constraint language", Version 2.0, 2006.

[30] L. C. Briand, S. Morasca, and V. R. Basili, "Property-Based Software-Engineering Measurement", IEEE Transactions on Software Engineering, Vol. 22, No. 1, 1996.

[31] OMG, "MDA guide", Version 1.0.1, 2003.

[32] W3C, "Web services description language (WDSL)", Version 1.1, 2001.

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