Participatory Service Design through Composed and Coordinated Service Feature Models

Erik Wittern, Nelly Schuster, Jörn Kuhlenkamp, and Stefan Tai

eOrganization Research Group, Karlsruhe Institute of Technology
Englerstr. 11, 76131 Karlsruhe, Germany
{Erik.Wittern,Nelly.Schuster,Joern.Kuhlenkamp,Stefan.Tai}@kit.edu
http://www.eOrganization.de

Abstract. Active participation of diverse stakeholders such as consumers or experts in service engineering is critical. It ensures that relevant aspects of service quality, service acceptance and service compliance are addressed. However, coordination of diverse stakeholder inputs is difficult and their collaborative creation of common design artifacts demands novel engineering solutions. We present a service-oriented approach for engineering design artifacts: service feature models are introduced as compositions of model parts that can be contributed by different stakeholders and software resources acting as services. Our method and tool applies service-orientation to collaborative design, thereby taking participatory service engineering to the level of coordinated service composition.

Keywords: Service engineering, service feature modeling, coordination, collaboration, participatory service design.

1 Introduction

Participatory service engineering is about the involvement of different stakeholders, in the role of service providers and service consumers, into the analysis, design, and development of services [10,11]. In the public sector, for instance, stakeholders include citizens, municipalities, and corporations. Involving different stakeholders into the service engineering life cycle is a non-trivial task; however, participation promises to better meet the interests and needs of all parties involved, to improve customer satisfaction, and to better comply with relevant policies and laws [10].

In service design, stakeholders typically are represented by groups of experts, including software engineers, infrastructure providers, decision-makers, and legal experts. These stakeholders collaborate with each other, contributing specific knowledge. Results of the collaboration are manifested in one or more design artifacts (such as documents or code), which correspondingly address the diversity of relevant service aspects, including technical, business-related or legal ones.

Participatory service design can thus be seen as the process of coordinating a set of stakeholders, where each stakeholder is represented by one or more experts

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 158–172, 2012.

and contributes to the creation of design artifacts. Here, we introduce *service feature models (SFMs)* as a modeling approach for service design to create such coordinated, composed design artifacts: SFMs capture design aspects that are contributed by different stakeholders (or even other sources). SFMs include design alternatives (e.g. the set of authentication mechanisms the implementation of a service could use), decisions (e.g. which authentication mechanism to use), and constraints (e.g. that a service has to be delivered electronically). They can be composed to serve as a single design artifact that describes diverse relevant design issues.

This design artifact is a document composed of services: model parts are contributed by human-provided services or by software services. Thus, responsibilities for the specification of design aspects can be delegated to dedicated experts who can work on them in parallel. In addition, data from software services or form resources on the Web can be integrated, which can serve as a base for design decision-making. To manage causal dependencies between model parts and to govern the collaborative composition activities, we define coordination mechanisms that involve a coordinator role along with a set of lightweight coordination rules.

The remainder of this paper is structured as follows: In Section 2 we introduce and discuss service feature modeling as a technique to support service design. We present its methodology in service design, its modeling elements and shortly discuss how we used the approach in the context of the COCKPIT project [6]. In Section 3, we present our approach to compose SFMs of services. For this purpose, we build on our composition and coordination model for document-based collaboration [16] and define (a) a *service composition model* which allows coordinators to delegate modeling tasks to responsible experts based on the modularization of SFMs and (b) a set of *coordination rules* which allow the (semi-)automated management of dependencies between different model parts and activities. The system architecture and proof-of-concept implementation that we used to determine the applicability of our approach is described in Section 4. In Section 5, we discuss related work. Finally, we summarize our work and present an outlook in Section 6.

2 Service Feature Models

To provide a comprehensive view of a participatory service design, design artifacts stemming from experts of different domains or disciplines need to be captured and integrated in one service model. *Service feature modeling* has been designed for this purpose and has been conceptualized in previous work [19]. Service feature modeling builds upon the well-established feature modeling approach from software product line engineering [12]. Therefore, SFMs benefit from efficient modeling methodologies, formalisms and automatic reasoning capabilities designed for standard feature models, e.g., [4,3,7,8,13].

2.1 Methodology

Iterative service design and rapid development of service design alternatives allow to communicate, discuss and assess design alternatives, improving overall service quality. We distinguish two phases in iterative service design: first, a rapid *modeling phase*, second, a detailed *configuration phase*.

In the modeling phase, a SFM is created as a design artifact that represents a number of relevant design aspects. Multiple design alternatives are modeled in this design artifact that represents a set of possible implementations of the service.

In the configuration phase, decisions between design alternatives are made. The resulting artifact of the configuration phase is a *configuration*. In contrast to a SFM, a configuration represents only a single valid service design. A configuration serves as input for service implementation and documents the design decisions made. During the configuration phase, constraints and preferences are specified based on requirements. Requirements are collected from stakeholders, for instance citizens in public service design. Modeling of constraints and preferences improves service design because it reveals trade-offs that result from choosing one design alternative over another.

2.2 Modeling Elements

An exemplary SFM is illustrated in Figure 1. It represents (simplified) the "Access extracts of insurance record in Social Security Organization (GR01)" service that the Greek Ministry of Interior provided as a service scenario in the COCKPIT project. The service's goal is to allow employees to access their insurance record to verify that their employers contributed to the social security insurance.

The *service feature diagram* is the graphical representation of a SFM, in which *features* are decomposed into a tree-structure. Features represent service design

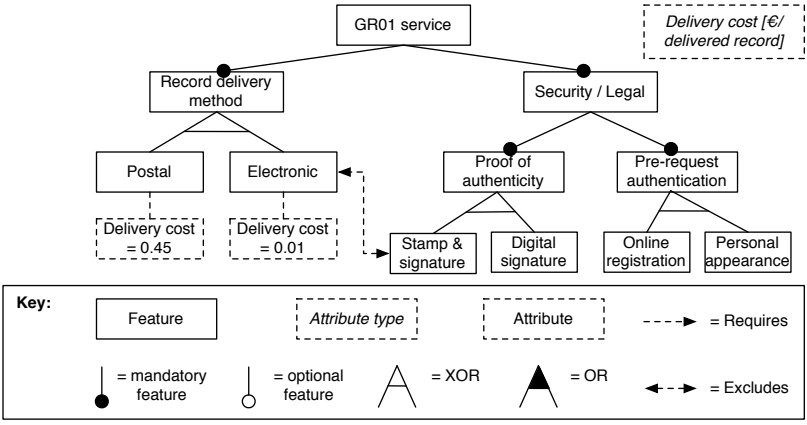

Fig. 1. Exemplary "Access extracts of insurance record in Social Security Organization (GR01)" service modeled (simplified) as SFM

aspects of relevance for a stakeholder, e.g. an access channel, a work-flow variant or a legal aspect. Features can be denoted as *mandatory* or *optional*. This allows service engineers to derive several configurations during the configuration phase. Features can also be grouped in *XOR* or *OR* groups, meaning that for a valid configuration, exactly one or at least one feature from the group needs to be selected. Causal dependencies can be specified through *cross-tree relationships*. They depict that one feature either *requires* or *excludes* another feature to be part of a valid configuration. For example, in Figure 1, if "stamp & signature" is selected, the record delivery method "electronic" cannot be selected and vice versa. *Attributes* describe measurable, non-functional characteristics of features [4]. For example, an attribute can denote the "delivery cost" of a feature representing the delivery method. *Attribute types* capture common characteristics of attributes. For example, in Figure 1, the attribute type "delivery cost" defines the measure for all attributes of that type to be "€/ delivered record".

2.3 Service Feature Modeling in Practice

The COCKPIT EU project aims to enable diverse stakeholders, including e.g. citizens, to participate in the design of public services [6]. We developed service feature modeling in this context to address the necessity of combining diverse service design aspects in a single model. In COCKPIT, service feature modeling was applied to three public service (re-) design scenarios. SFMs were modeled that represent multiple variants of how to provide the public service in question. During the configuration phase, citizens participated by stating their preferences regarding the SFM's attribute types on a deliberation platform. These preferences were, using multi-criteria decision making, applied to the possible service alternatives to determine a ranking of the alternatives. However, we learned in COCKPIT that it is highly desirable not only to enable participation in the configuration but also in the modeling of SFMs to integrate expert knowledge from diverse disciplines at an early stage. This is what motivated our approach for composed and coordinated SFMs.

3 Collaborative Service Feature Modeling

In collaborative service feature modeling, model parts contributed by diverse participants are composed into a single SFM. The structure of the composition is defined by our *service composition model*. *Roles* describe activities that participants perform. *Coordination mechanisms* provide the required coordination of the service composition and modeling activities.

Figure 2 depicts an example where several participants collaborate to create the "GR01" service. As illustrated, responsibilities for the modeling of certain service aspects are separated among several participants: a "service engineer", responsible for the creation of the overall service design, contributes an initial SFM representing the "GR01" service. For the aspect "security / legal", a dedicated SFM is delivered by a "legal expert". On the lower left side, a "cost estimation service" contributes attribute values for two attributes of the type "delivery

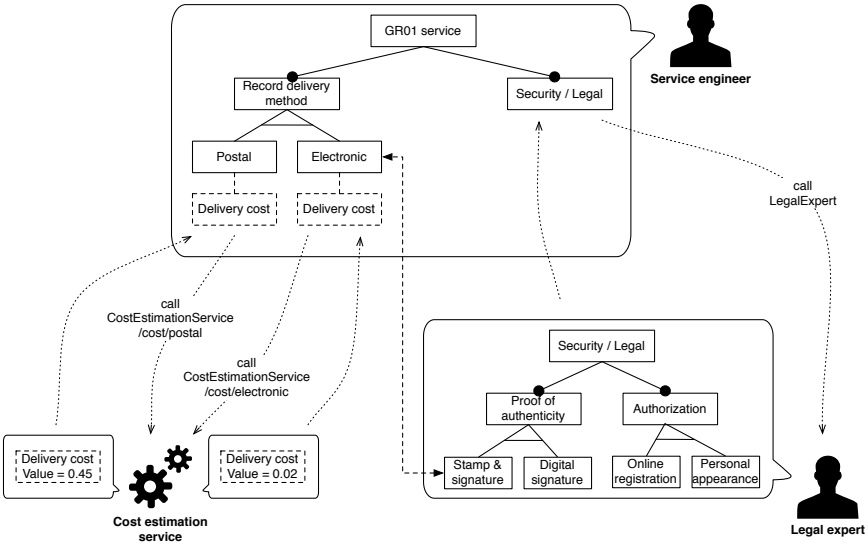

Fig. 2. Example of SFM composed of services

cost". The overall SFM representing the "GR01" service is thus a composition of results contributed by human and non-human service providers.

3.1 Service Composition Model

The service composition model defines how results and services can be composed in order to derive a coherent SFM. The model is illustrated in Figure 3. It is based on and extends previous work [16,17]. Note that *service* does not denote the service to be modeled but the services that contribute results.

The intended outcome of the collaboration, a composed SFM, consists of multiple *results*. We distinguish two specific types of (expected) results:

– **SFMs** are parts of a larger SFM. The *sub result* relationship denotes the potential nesting of SFMs. For example, in Figure 2, the SFM corresponding to the "GR01" service contains another SFM corresponding to the service's "security / legal" aspects. The tree structure of the SFM itself does not

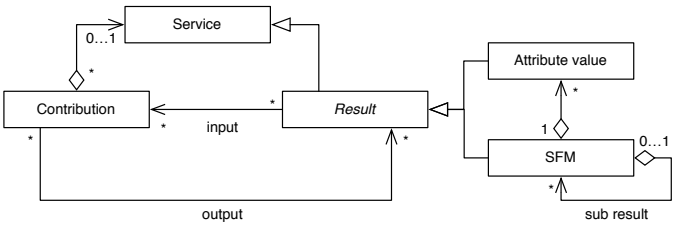

Fig. 3. Service composition model, based on [16]

necessarily correspond to the tree structure of the service composition model. As defined in Section 2.2, SFMs contain features, attribute types, attributes and cross-tree relationships.
- **Attribute values** are primitive data types (i.e. integer, double or string) representing measurable, non-functional characteristics of a feature. As such, attribute values cannot, in contrast to SFMs, be decomposed further into sub results. In Figure 2, two attribute values are contributed to the "delivery cost" attributes of the "postal" and "electronic" delivery.

Contributions denote the delivery or transformation of results. Results are the output of one or more contributions. A contribution is performed by a dedicated human-based or software service. This service may transform existing results which it gets as input. Separating contributions from results provides the flexibility of having multiple contributions collectively work on a result or of using a single contribution for delivering or transforming multiple results. For example, for a SFM the contributions "create SFM", "validate SFM" and "approve SFM" may exist. In Figure 2, contributions are illustrated as speech bubbles delivered by the services "service engineer", "cost estimation service" and "legal expert". Results themselves are offered as services and can be reused in other SFM modeling projects.

3.2 Roles

During collaboration, participants take different roles. A role describes a set of activities an actor may perform in participatory service design. A single actor may engage in multiple roles at the same time or vary roles over time [18]. We refer to an actor in one of the three following roles by the name of the role:

- **Modelers** contribute SFMs. They define how to decompose features, which attributes to use to describe properties of features and what attribute types they relate to, and finally what cross-tree relationships exist between features. In general, we assume that human actors hold the role modeler (e.g. a legal expert).
- **Attribute value providers** provide contributions to attribute values. To account for dynamic changes in attribute values during and after service design as well as for complex calculations of attribute values, we propose to utilize non-human actors, e.g. Web services, for the role of an attribute value provider (e.g. a benchmark service).
- **Coordinators** have two assigned responsibilities. Firstly, they identify contributions and assign modeling tasks for both SFMs and attribute values. Secondly, coordinators assign services to identified contributions in the role of either modeler or attribute value provider. Coordination activities can be delegated by assigning the coordinator role with respect to a single result. This allows for coordination activities within a different department involved in the collaborative service design (e.g. the legal department).

Modeler and coordinator roles are closely related because the division of labor is performed using the overall SFM structure and requires basic understanding of modeling.

3.3 Coordination Mechanisms for Service Feature Modeling

During service feature modeling, human experts collaborate to construct the SFM according to the component model described in Section 3.1. To start the collaboration, a coordinator defines several required results and contributions. In order to delegate tasks, the coordinator assigns human experts or software services to the contributions. During collaboration, additional results or contributions are added or services assigned. As soon as services are assigned, they are allowed to participate through delivering or transforming respective results.

The challenge in realizing such collaboration is to coordinate interactions of services with the overall SFM in order to avoid inconsistencies due to dependencies between tasks or contributions, e.g., due to cross-tree relationships. To coordinate collaborative service feature modeling, we identified four required types of *coordination mechanisms*.

Coordination of Basic Interactions. In our previous work, we specified and visualized a *service binding protocol* for service assignments [16] allowing the delegation of results to responsible modelers. This protocol is suitable for SFM compositions as well: The binding of a service to a contribution is initiated by the coordinator, asking the service for its commitment to contribute. The service may accept or decline. We expect that, in most cases, the service provider will accept the binding as he was selected based on his expertise for the respective service model part. Software services automatically accept a binding request.

As soon as a service is bound, it can deliver the results related to the assigned contribution, e.g., new SFMs or attribute values. The response of the service is an update of the associated output results. In addition, during collaboration, a bound service can be manually called by a coordinator, for instance, as a reminder to contribute not yet delivered results. The results can be updated by the service until they are approved by a coordinator. This is defined in the *service request / response protocol* [16].

These protocols allow easy integration of human and software services alike as they do not require complex activities. All service providers need to implement the protocols. This hierarchical coordination model is suitable for multidisciplinary service design projects, where experts for different domains as well as responsible service engineers for a SFM exist that are responsible for the delegation of work to experts.

Coordination of Cross-Tree Relationships. Cross-tree relationships denote that one feature either requires or excludes the existence of another feature in a configuration (see Section 2.2). Changing or deleting a feature that is part of a cross-tree relationship can cause inconsistencies in a SFM. Thus, if a cross-tree relationship in a result relates to a feature in another result, a notification to the modeler of the cross-tree relationship should be sent if the feature is changed or deleted.

For the management of such dependencies, we use the event-condition-action (ECA) rule mechanism of [16] and configure it with event types and rules required

for collaborative service feature modeling. Events are emitted on changes of results as well as on each transition in the basic interaction protocols presented above. Events are input to rules which might trigger actions, e.g. service calls or the sending of notification messages.

To coordinate cross-tree relationships, we define the events *FeatureUpdated* and *FeatureDeleted* that are emitted on changing or deleting an SFM containing the feature. Additionally, if a cross-tree relationship is specified by a modeler in an SFM and the SFM is updated, a rule is automatically instantiated that triggers adequate measures in reaction to these events. For example, based on the SFM illustrated in Figure 2, as soon as the legal expert identifies the exclude relationship between delivery method "electronic"and "stamp & signature", a rule is created as defined below. If "electronic" is changed to "certified mail", the legal expert is notified to check the excludes dependency to "stamp & signature" for validity and possibly adapt it. (Note: the values "electronic" and "legal expert" in the following rule denote unique, non-changeable IDs of the results or contributions, respectively.)

```
EVERY FeatureUpdated("electronic") OR FeatureDeleted("electronic")
DO notify("legal expert");
```

Coordination of Attribute Type Dependencies. Attribute types allow modelers to define common characteristics of attributes, e.g. the measurement unit for "cost". If an attribute type is changed or deleted, potential inconsistencies can occur with regard to the attributes relating to this type. Thus, a modeler of an attribute should be notified in such cases. Similar to handling cross-tree relationships, attribute type dependencies are coordinated using ECA rules. The dedicated events are *AttributeTypeUpdated* and *AttributeTypeDeleted*. Rules for such attribute type dependencies listen for events denoting a change of the attribute type. The rules are automatically created as soon as an SFM is created or updated including an attribute that relates to an attribute type in another result. As action, notifications are sent to the modelers of SFMs including attributes of this type. For instance, if the measurement unit of an attribute type "delivery cost" is changed from "€/ delivered record" to "€/ month", the modelers of SFMs including an attribute using this attribute type are notified and possibly request the according attribute value providers with updated parameters.

```
EVERY AttributeTypeUpdated("delivery cost")
   OR AttributeTypeDeleted("delivery cost")
DO notify("service engineer")
```

Coordination of Attribute Provisioning Dependencies. Contributions of attribute value providers are useful to include real-time data into the SFM or include complex data that results from a calculation. The invocation of such services should be performed during the configuration phase since currentness of attribute values influences the selection among alternatives. As we can not always count on the automated push of attribute values from external Web services in case they have recent data, we suggest to define new event types

which denote the end of the modeling phase and are emitted, for instance, if requested via the modeling tool. The event might then trigger a rule which requests all services delivering an attribute value. Additionally, a rule might request services at certain points in time, e.g., every morning at 8am. Such rules can be manually specified throughout the collaboration by the coordinators of SFMs. Alternatively, we suggest that they are created as soon as attribute value providers are bound to a contribution and deleted if the binding is removed.

```
EVERY ModelingPhaseFinished
DO requestContribution("big machine benchmark");

AT (timer:8am)
DO requestContribution("big machine benchmark");
```

We believe that the proposed event types and rules are suitable to avoid inconsistencies during service feature modeling. However, we do not claim completeness. As the ECA rules coordination approach is very generic and extensible, additional rules and event types can be specified whenever needed. For example, modelers can specify rules which notify them if a certain result was delivered or updated, using events of type *SFMUpdated* or *SFMDeleted*.

4 Proof of Concept

We designed a system architecture supporting the conceptualized collaboration model and prototypically implemented it. The implementation strongly builds upon and extends the implementation used within the COCKPIT EU project [20].

4.1 Architecture and Components

The architecture is shown in Figure 4. The individual components are presented in the following subsections.

Service Feature Model Designer (SFM Designer): The *SFM designer* provides a user interface for participating experts. It addresses two basic functionalities. First, it provides *modeling* capabilities via graphical UI used to create and adapt SFMs that are provided as results. As such, it acts as a service adapter for these human experts who, as modelers, contribute results. Second, the SFM designer provides *coordination* capabilities. It allows coordinators to define (expected) results and to define which services should contribute the results. Supporting the configuration phase, the SFM designer can determine all possible configurations for a given SFM and aggregates their attributes.

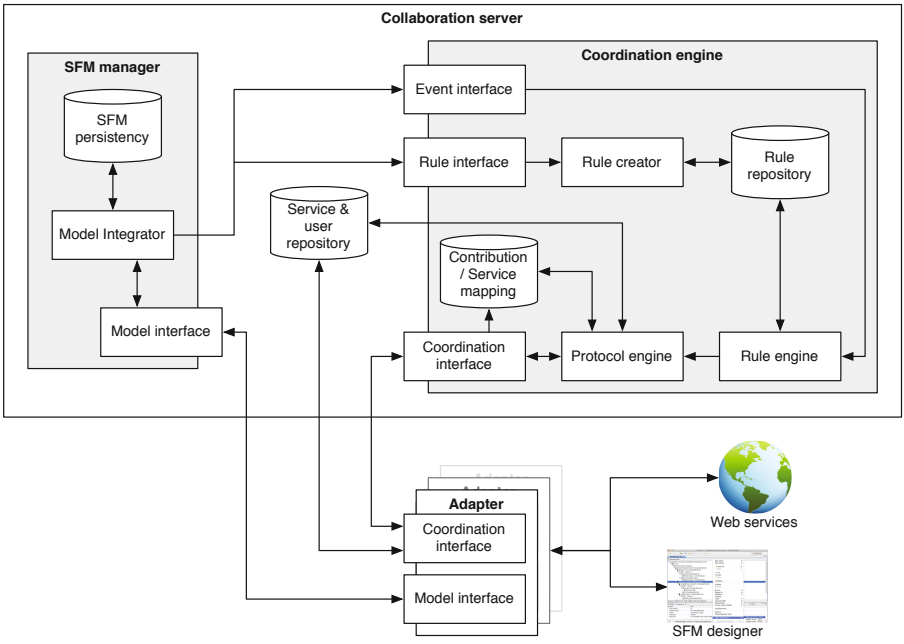

Fig. 4. Architecture of a system for collaborative service feature modeling

Adapters: Our system design foresees numerous adapters that allow services to participate in the collaboration. Adapters ensure compatibility of the service interfaces and our system's interfaces, e.g. they implement the coordination protocols described in Section 3.3. Per service interface, a dedicated adapter is required. Adapters have two interfaces to communicate with our system: via the *coordination interface*, services can be asked for binding and can then be requested to update according results. The *model interface* is used to retrieve existing results of the model in focus and to contribute (create, update, or delete) results.

Collaboration Server: The *collaboration server* handles the coordination required for asynchronous collaboration in service feature modeling. It consists of three components, the *service & user repository*, the *SFM manager* and the *collaboration engine*.

The SFM manager stores the contributed results, namely SFMs and attribute values, in the *SFM persistency* component. Using the *model interface*, any service bound via adapters can create, retrieve, update or delete results - thus, for both SFM and attribute value results, CRUD methods are provided. Results sent or requested pass through the *model integrator*. It checks committed results for a) model elements that require coordination rules to be defined, e.g. attributes relating to attribute types outside of the result, and b) changes w.r.t. model elements that require coordination, e.g. changes to cross-tree relationships. In such cases, the model integrator triggers the *coordination engine* to create rules or throw events. Further, if a result from the collaboration server is requested, the model integrator composes it by integrating all sub results into one coherent SFM.

The coordination engine contains the coordination logic. The *coordination interface* allows services via adapters to participate in the coordination. A coordinator can consult the *service & user repository* to find an adequate service to associate with a contribution. The association is stored in the *contribution / service mapping*. The *protocol engine* controls the binding and the service request / response protocol of the service based on information found in both the contribution / service mapping and the service & user repository. The *rule interface* triggers the *rule creator* when new model elements are contributed that require creation of a new coordination rule. Additionally, it can be used by any coordinator to manually define rules. Rules are stored in the *rule repository*. Through the *event interface* events can be sent to the *rule engine*. On receiving an event the rule engine checks existing rules and possibly triggers an action. For example, if an attribute type is changed, the "AttributeTypeUpdated" event triggers a previously specified rule which notifies all depending modelers. Notifications are sent via the *protocol engine* that communicates via the coordination interface with the respective service adapters.

4.2 Implementation

We implemented the SFM designer as an Eclipse-based rich client on top of the Eclipse Modeling Framework (EMF)[1]. Adapters are built on RESTful design principles utilizing Jersey[2]. Figure 5 shows the user interface of the SFM designer. We implemented our collaboration server based on the Grails Web

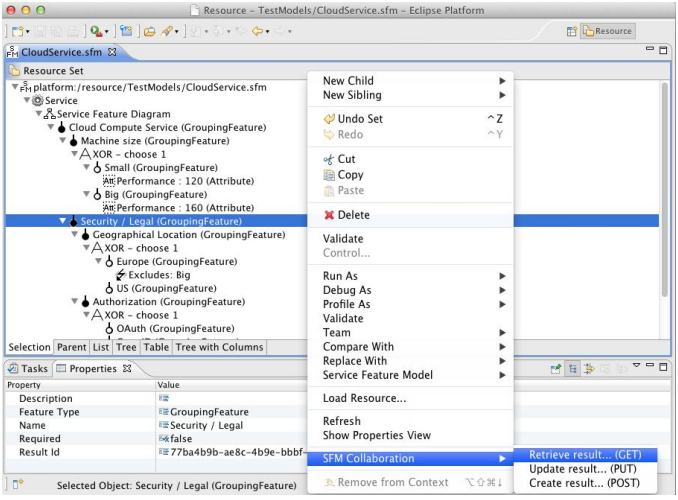

Fig. 5. Screen shot of the SFM designer with exemplary SFM and menu bar entries allowing to coordinate collaboration

[1] http://www.eclipse.org/modeling/emf/
[2] http://jersey.java.net/

application framework[3] and RESTful design principles. We built Java and Groovy[4] server libraries. The SFM manager stores SFMs within the SFM persistency in XML format. Service repository, rule repository and contribution/service mapping persist data in a MySQL database. We use ESPER[5] to implement our rule engine. Rules are expressed in the ESPER Event Processing Language (EPL).

Using our prototype implementation, we asserted the functionality of the presented approach. We are able to successfully assign contributions to human-based and Web services. For SFM results, corresponding resources are created, updated, retrieved and deleted by the SFM manager. On updating SFMs, the model integrator triggers the creation of rules and triggers events w.r.t. existing rules as conceptualized in Section 3.3. Notifications are currently send via e-mail. Alternative reactions depend on the implementation and can include, for instance, triggering service calls. Our implementation overall shows the applicability of our approach to enable collaborative creation of SFMs composed of services and thus affirms its ability to foster participatory service design.

5 Related Work

Some approaches to participatory service design exist. For example, [10] presents a high-level methodology for the participatory analysis, design, and optimization of services in the public sector. Our approach is in-line with this methodology and similar works and extends them by contributing a specific modeling notation and corresponding tools.

Regarding collaborative modeling, most of the approaches we found discuss the creation of models in the CAD domain where several experts work together to derive a graphical model of a product. For instance, the authors of [5] present an approach for collaborative editing of a central model maintained on a server, also addressing basic coordination problems, e.g. concurrency and synchronization. They do not consider a human coordinator or the creation of model parts by services. In contrast, we aim to allow a coordinator to split the model into parts to be delegated to responsible experts. We thus provide coordination mechanism on an application level.

Several works address how multiple *feature models* can be combined. [1] proposes to compose feature models that address specific domains, aiming to better deal with rising complexity for large feature models, to foster the model's evolution and also to engage diverse stakeholders into the modeling. In [2], a representation of feature models using description logics and a corresponding configuration method is presented to allow multiple experts to model and configure feature models. Both works focus on how to combine multiple models but do not address the coordinated creation of models or the integration of up-to-date values. Methodologies addressing the modeling of modular feature models

[3] `http://www.grails.org`
[4] `http://groovy.codehaus.org/`
[5] `http://esper.codehaus.org/`

are named as an intended future work. In contrast, we focus on the coordination of creating modular feature models collaboratively.

The approach presented in [9] focuses on collaborative modeling in software engineering. It allows software engineers to decompose UML diagrams into fine-grained design model parts that can be modified by distributed participants. The approach has some similarities to our approach, e.g., hierarchically breakdown of models into parts, event-based notifications and coordination mechanisms to manage concurrent access and dependencies between model parts. In [22] a model and tool are presented that enable software architects to collaboratively capture architectural decision alternatives and decision outcomes in a specialized Wiki. In the modeling phase, architects can define dependencies between decisions. Alternatives are used to ensure consistent and correct decision-making during the configuration phase. Despite some similarities, both presented approaches do not (yet) support delegation of modeling parts through a coordinator and do not enable the integration of content provided by software services into the models.

Flexible composition of services through end-users has been discussed in the mashups area [21]. Mashups allow end-users to easily compose and integrate (Web) services into artifacts. In addition, approaches for the integration of human-provided services into collaboration exist [15]. However, we are not aware of any approach that allows participants to create models through a mashup mechanism.

Overall, having analyzed related work in various research areas, we believe that our approach uniquely combines coordination and service-composition concepts to support participation of various experts in service design.

6 Conclusion and Future Work

In this paper, we addressed participatory service design by presenting SFMs as design artifacts capable of integrating various design issues. As we experienced in the COCKPIT project, especially the modeling phase in the SFM methodology allows for discussion and knowledge exchange in an early stage of the service design. Accordingly, we presented a service-composition model which allows participants to delegate responsibilities for model parts to experts who can independently contribute their parts to a central, uniform design artifact. By separating responsibilities based on the model structure, we tackle the challenge of integrating sub-models into a coherent model while still allowing participants to model in parallel [14]. In our approach participants act as human or software services, allowing the integration of dynamic or complex data into SFMs which can be kept up to date automatically. Collaboration activities are coordinated through a) the delegation of work based on the SFM structure and b) interaction protocols and a simple event-condition-action rule mechanism. This mechanism can also be used as a notification mechanism to manage causal dependencies between model parts, for instance cross-tree relationships. We presented the architecture of a system realizing our approach and a proof-of-concept implementation that allowed us to act out the relevant use cases of our approach.

In future work, we plan to address the collaborative configuration of SFMs. Configuration results could be used to incorporate the configuration of a (sub) SFM. It may be noted, however, that the collaborative configuration of feature models is already addressed in numerous works, e.g. [7,13]. Further, we want to investigate means to support consensus-based decisions on model parts, e.g. quorum-based decisions. We also envision the parameterization of service calls from a SFM based on dependent results, which would allow for even more flexible integration of results.

Acknowledgment. This work was supported by the COCKPIT project [6].

References

1. Acher, M., Collet, P., Lahire, P., France, R.: Composing Feature Models. In: van den Brand, M., Gašević, D., Gray, J. (eds.) SLE 2009. LNCS, vol. 5969, pp. 62–81. Springer, Heidelberg (2010)
2. Bagheri, E., Ensan, F., Gasevic, D., Boskovic, M.: Modular Feature Models: Representation and Configuration. Journal of Research and Practice in Information Technology 43(2), 109–140 (2011)
3. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated Analysis of Feature Models 20 Years Later: A Literature Review. Information Systems 35(6), 615–636 (2010)
4. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated Reasoning on Feature Models. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 491–503. Springer, Heidelberg (2005)
5. Bidarra, R., Berg, E.V.D., Bronsvoort, W.F.: Collaborative Modeling with Features. In: Proc. of the 2001 ASME Design Engineering Technical Conferences, DETC 2001, Pittsburgh, Pennsylvania (2001)
6. COCKPIT Project: Citizens Collaboration and Co-Creation in Public Service Delivery (March 2012), `http://www.cockpit-project.eu`
7. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing Cardinality-based Feature Models and their Specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
8. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration through Specialization and Multilevel Configuration of Feature Models. Software Process: Improvement and Practice 10(2), 143–169 (2005)
9. De Lucia, A., Fasano, F., Scanniello, G., Tortora, G.: Enhancing collaborative synchronous UML modelling with fine-grained versioning of software artefacts. Journal of Visual Languages and Computing 18(5), 492–503 (2007)
10. Hartman, A., Jain, A.N., Ramanathan, J., Ramfos, A., Van der Heuvel, W.-J., Zirpins, C., Tai, S., Charalabidis, Y., Pasic, A., Johannessen, T., Grønsund, T.: Participatory Design of Public Sector Services. In: Andersen, K.N., Francesconi, E., Grönlund, Å., van Engers, T.M. (eds.) EGOVIS 2010. LNCS, vol. 6267, pp. 219–233. Springer, Heidelberg (2010)
11. Holmlid, S.: Participative, co-operative, emancipatory: From participatory design to service design. In: 1st Nordic Conference on Service Design and Service, Oslo, Norway (2009)
12. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. rep., Carnegie Mellon University (November 1990)

13. Mendonça, M., Cowan, D., Malyk, W., Oliveira, T.: Collaborative Product Configuration: Formalization and Efficient Algorithms for Dependency Analysis. Journal of Software 3(2) (2008)
14. Renger, M., Kolfschoten, G.L., de Vreede, G.J.: Challenges in Collaborative Modeling: A Literature Review. In: Dietz, J.L.G., Albani, A., Barjis, J. (eds.) CIAO! 2008 and EOMAS 2008. LNBIP, vol. 10, pp. 61–77. Springer, Heidelberg (2008)
15. Schall, D., Truong, H.L., Dustdar, S.: Unifying human and software services in web-scale collaborations. IEEE Internet Computing 12(3), 62–68 (2008)
16. Schuster, N., Zirpins, C., Scholten, U.: How to Balance between Flexibility and Coordination? Model and Architecture for Document-based Collaboration on the Web. In: Proc. on the 2011 IEEE Int. Conf. on Service-Oriented Computing and Applications (SOCA), pp. 1–9 (2011)
17. Schuster, N., Zirpins, C., Tai, S., Battle, S., Heuer, N.: A Service-Oriented Approach to Document-Centric Situational Collaboration Processes. In: Proc. of the 18th IEEE Int. Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2009, pp. 221–226. IEEE Computer Society, Washington, DC (2009)
18. Sonnenwald, D.H.: Communication roles that support collaboration during the design process. Design Studies 17(3), 277–301 (1996)
19. Wittern, E., Zirpins, C.: On the Use of Feature Models for Service Design: The Case of Value Representation. In: Cezon, M., Wolfsthal, Y. (eds.) ServiceWave 2010 Workshops. LNCS, vol. 6569, pp. 110–118. Springer, Heidelberg (2011)
20. Wittern, E., Zirpins, C., Rajshree, N., Jain, A.N., Spais, I., Giannakakis, K.: A Tool Suite to Model Service Variability and Resolve It Based on Stakeholder Preferences. In: Pallis, G., Jmaiel, M., Charfi, A., Graupner, S., Karabulut, Y., Guinea, S., Rosenberg, F., Sheng, Q.Z., Pautasso, C., Ben Mokhtar, S. (eds.) ICSOC 2011. LNCS, vol. 7221, pp. 250–251. Springer, Heidelberg (2012)
21. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development. IEEE Internet Computing 12(5), 44–52 (2008)
22. Zimmermann, O., Koehler, J., Leymann, F., Polley, R., Schuster, N.: Managing Architectural Decision Models with Dependency Relations, Integrity Constraints, and Production Rules. Journal of Systems and Software 82(8), 1249–1267 (2009)