Cloud Service Selection Based on Variability Modeling

Erik Wittern[1], Jörn Kuhlenkamp[1], and Michael Menzel[2]

[1] eOrganization Research Group, Karlsruhe Institute of Technology (KIT)
Englerstr. 11, 76131 Karlsruhe, Germany
{Erik.Wittern,Joern.Kuhlenkamp}@kit.edu
http://www.eorganization.de

[2] Research Center for Information Technology
Karlsruhe Institute of Technology (KIT)
Menzel@fzi.de

Abstract. The selection among Cloud services is a recent problem in research and practice. The diversity of decision-relevant criteria, configurability of Cloud services and the need to involve human decision-makers require holistic support through models, methodologies and tools. Existing Cloud service selection approaches do not address all stated difficulties at the same time. We present an approach to capture capabilities of Cloud services and requirements using variability modeling. We use *Cloud feature models* (CFMs) as a representation mechanism and describe how they are utilized for requirements elicitation and filtering within a presented *Cloud service selection process* (CSSP) that includes human decision-makers. Filtering produces a reduced number of valid Cloud service configurations that can be further assessed with current multi-criteria decision making-based selection approaches. We present software tools that we use to demonstrate the applicability of our approach in a use case about selecting among Cloud storage services.

Keywords: Cloud service selection, variability modeling, feature modeling, decision-making.

1 Introduction

Since the dawn of service computing, the problem of how to select software services is omnipresent for IT decision-makers. Service selection, typically, builds upon a) the representation of decision-relevant capabilities of the *candidates* to select among and b) the representation of the *requirements* and *preferences* of the decision-maker (e.g. a person, institution or automated agent). Both representations are matched to determine the candidate that best fulfills the decision-maker's needs. For example, in service-oriented computing, the selection of *Web services* has been addressed similarly: In policy matchmaking, policies capture capabilities and requirements towards Web services and are matched to determine a service to fulfill the request, e.g., in [17].

Recently, the growing number of *Cloud services* raises the need for dedicated representation of their capabilities as well as requirements and preferences towards them and for corresponding selection methods. Such approaches need to consider the specifics of selecting Cloud services: Cloud services typically hold state, for example, a Cloud storage service persists user data. Because of this and because Cloud service interfaces are not standardized, exchanging them is costly. Thus, Cloud service selection is a relatively long-lasting decision, compared, for example, with Web service selection. It must include strategic considerations like vendor lock-in or legal aspects that require involvement of human decision-makers. Cloud services, especially Infrastructure as a Service, frequently feature *configurability*: consumers can choose among many options on how to consume the service. For example, Amazon's *simple storage solution (S3)* allows consumers to define a preferred geographical location for the servers or to use different pricing schemes[1]. Overall, representations to support Cloud service selection need to be capable of representing the diverse decision-relevant aspects, must support integration of human decision-makers and must reflect Cloud services' configurability.

In this paper, we contribute to Cloud service selection in two ways: First, we present an approach to improve the way that Cloud service capabilities and consumer requirements towards them are represented. We introduce *Cloud feature modeling*, based on variability modeling, to address the presented challenges of representing Cloud services. Second, utilizing *Cloud feature models (CFMs)*, we present a *Cloud service selection process (CSSP)* as a methodology for decision-making. It includes narrowing down the number of service candidates based on stated requirements. Enabling automated support for delimiting the number of candidates is required to keep following decision phases manageable. The process also encompasses selection based on preferences that can make use of previously introduced multi-criteria decision making approaches, e.g. [10].

The remainder of this paper is structured as follows: Section 2 discusses related work from the area of Cloud service selection. Section 3 introduces our approach to model the Cloud service selection problem with Cloud feature modeling. We provide a formalization of the introduced modeling elements to clearly define their usage. Section 4 presents our CSSP, addressing the involved roles and activities. Section 5 presents our prototypical implementation of a modeling tool supporting our approach and discusses a use case to illustrate the approach's applicability. Finally, Section 6 discusses our work and gives an outlook on future research.

2 Related Work

Cloud service selection has recently been addressed in numerous publications: in [12], the authors propose a mechanism to automatically select Cloud storage services. Capabilities of candidate services are expressed in an XML schema and matched with requirements. The usage of *multi-criteria decision making*

[1] `http://aws.amazon.com/de/ec2/`

(MCDM) approaches for the selection of Cloud services has been proposed in multiple papers. In [13], the authors use a combination of a revised wide-band Delphi method to weight attributes and simple additive weighting to determine candidate ranking. In [7] and [10], the authors utilize Analytical Hierarchy Process (AHP) or Analytical Network Process (ANP) to support the selection. The presented hierarchies are designed to address aspects relevant to specific classes of Cloud services, i.e., IaaS, PaaS and SaaS. In [11], the authors present an overview of existing Cloud service selection approaches. They formalize the Cloud service selection problem and propose an approach to determine the similarities between a requirement vector and all candidates' capability vectors to recommend the most suitable candidate.

The presented approaches face certain limitations that we address in this paper. [12] omit support for XML definitions making the approach rather arduous to human decision-makers. It is commonly assumed, e.g. in [13,9,10,7], that representations of the service candidates to choose from are provided as input for the presented MCDM approaches. However, it is not discussed how to derive these representations. In contrast, we present a modeling approach and corresponding methodology to create such representations. The *configurability* that today's Cloud services offer is neglected in existing approaches that focus only on a handful of candidates. By considering configurability, we provide decision-makers with a decision-basis that better reflects actual choices. Many MCDM approaches also base their selection recommendation on preferences only, e.g. [7,10]. In contrast, we consider both requirements and preferences in our CSSP.

Overall, our approach addresses modeling of Cloud service candidates and a holistic approach for selecting among them - both aspects have not yet been addressed in literature.

3 Modeling the Cloud Service Selection Problem

We propose to utilize *variability modeling* as a foundation to model decision-relevant criteria of Cloud services. Variability modeling approaches, such as *feature modeling* [3], are commonly used to capture the commonalities and differences in system families. They enable to represent multiple *configurations* of a system in a single model. Within this section, we introduce our Cloud feature modeling approach.

3.1 Feature Modeling Basics

Our modeling approach builds upon *extended feature modeling* [3,2] to represent the commonalities and differences of Cloud services. We assume the definition of *feature* as a system property that is relevant to some stakeholder, be it a functional or non-functional property of the system [5]. Following this definition, we consider features to be on the right level of abstraction to capture aspects of value for service consumers [15] - and thus also to capture the diverse aspects relevant in Cloud service selection.

As a basis to describe our adaptations, we formalize feature modeling as follows: A *Feature Model (FM)* is represented by a directed graph $G = (V, E)$

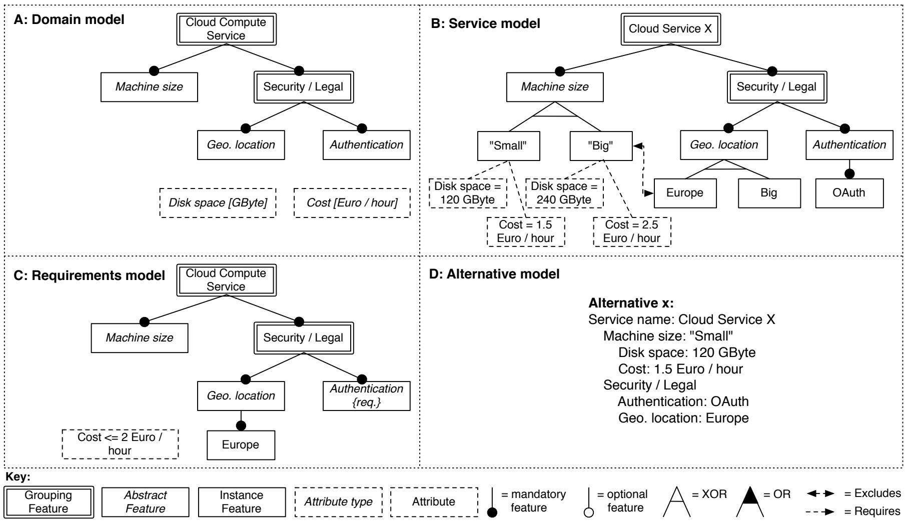
with the set of vertices V representing features and the set of edges E representing relationships between features. Each FM contains a single root feature $r \in V$. A relationship $e = \{init(e), ter(e)\}$ is described by the initial vertex $init(e) \in V$ and the terminal vertex $ter(e) \in V$. We distinguish between two types of relationships [8]: In *decomposition relationships* $E^{de} \subseteq E$, we refer to $init(e)$ as p_e and *parent feature* and to $ter(e)$ as c_e and *child feature*. The four sets $E^{mandatory}$, $E^{optional}$, E^{XOR}, $E^{OR} \subseteq E^{de}$ represent relationships that decompose features hierarchically. *Cross-tree relationships* $E^{cr} \subseteq E$ consist of the two sets $E^{requires}, E^{excludes} \subseteq E^{cr}$. They restrict the number of configurations represented by a FM. A *configuration* is a valid selection of features from a FM [2] that fulfills all specified relationships (e.g., a mandatory feature needs to be contained in each configuration where its parent feature is also contained). A feature is described by a number of *feature attributes* (FAs). We use the notation $v_i.x$ to refer to the feature attribute x of feature v_i. We capture FAs in a quantitative manner [3]. A quantitative approach enables automatic analysis upon FMs and to capture infinite domains in a clear and concise way [6]. According to [2] no consensus on a notation on FAs exists, but it is agreed on that FAs at least comprise of the basic building blocks *name*, *domain* and *value*. We refer to the basic building blocks by $name(v_i.x)$, $dom(v_i.x)$ and $val(v_i.x)$.

3.2 Cloud Feature Modeling

Despite the described potential of FMs for Cloud service selection (i.e. level of abstraction of features, capability to represent configurability), FMs were neither specifically designed for decision support nor to model Cloud services. Therefore, in this section, we describe how to adapt the model and notation of FMs to support our CSSP presented in Section 4. We refer to an adopted feature model as Cloud feature model (CFM). Figure 1 presents a very simple example of our approach's models and their elements for illustration purposes.

Enriched Feature Attributes. Feature attributes serve different purposes in our modeling approach: they describe quantitative and numeric properties of Cloud service offers and requirements of decision-makers regarding these properties. Feature attributes allow for an automated mapping of configurations to MCDM approaches.

Feature Attribute Types (FAT) are introduced to specify global information regarding multiple feature attributes of the same type. Using them avoids redundant specification of attribute information and allows to provide *standard aggregation strategies (SASs)* for feature attributes. FATs are referenced by the name of a FA. Therefore, two FAs $v_i.x$ and $v_j.x$ reference the same FAT if $name(v_i.x) = name(v_j.x)$ holds. Within a FAT at least the attribute domain and a SAS for a group of feature attributes is specified. We limit our approach to quantitative attribute values. Therefore, attribute domains can be discrete or continuous and finite or infinite. Examples for attribute domains are *integer* and *real*. A *boolean* domain may be represented by 0,1 [8].



Alternative x:
Service name: Cloud Service X
Machine size: "Small"
Disk space: 120 GByte
Cost: 1.5 Euro / hour
Security / Legal
Authentication: OAuth
Geo. location: Europe

Fig. 1. Exemplary models illustrating model types and modeling elements

Standard Aggregation Strategies (SAS) are used to aggregate all values of FAs with the same FAT for a single configuration. Within literature, approaches exist that aggregate feature attributes [3] or specify global constraints for aggregations of feature attribute values [8]. However, our approach takes an explicit process model, re-usability of models and multiple stakeholders into account. We define a SAS of a FAT abstractly as a closed (binary) function on the attribute domain specified within the FAT. Examples for commutative SASs on the attribute domain *integer* are functions that represent addition and multiplication.

Feature Types. We use feature types to represent additional design constraints, limiting the number of valid modeling choices, and provide additional semantics in contrast to traditional features. Therefore, a number of existing automated analysis approaches for FMs [2] are applicable for resulting CFMs, respectively. We propose to utilize *grouping features*, *abstract features* and *instance features.* Each feature type is represented by a set of features $V^{gr}, V^{ab}, V^{in} \subseteq V$. No additional feature types are included.

$$V \backslash (V^{gr} \cup V^{ab} \cup V^{in}) = \emptyset \tag{1}$$

Furthermore, each feature belongs to a single feature type.

$$V^{gr} \cap V^{ab} = V^{gr} \cap V^{in} = V^{ab} \cap V^{in} = \emptyset \tag{2}$$

Grouping Feature: Grouping features decompose the decision problem, thus helping to organize and structure the CFM. Grouping features contain a set of abstract features which address a similar concern, thus providing a comprehensive view for different stakeholders. For example, all abstract features concerning *security* can be grouped underneath an according grouping feature. The root feature of a CFM is a grouping feature.

$$r \in V^{gr} \tag{3}$$

The parent feature of a grouping feature must be a grouping feature.

$$c_{e_i} \in V^{gr} \rightarrow p_{e_i} \in V^{gr} \forall e_i \in E^{de} \tag{4}$$

Grouping features decompose to abstract features and grouping features and not to instance features.

$$p_{e_i} \in V^{gr} \rightarrow c_{e_i} \in (V^{gr} \cup V^{ab}) \forall e_i \in E^{de} \tag{5}$$

The purpose of grouping features is to solely structure abstract features. Grouping features are only decomposed by mandatory decomposition relationships. Therefore, grouping features do not add variability to a CFM.

$$p_{e_i} \in V^{gr} \rightarrow e_i \in E^{mandatory} \forall e_i \in E^{de} \tag{6}$$

Abstract Feature: An abstract feature defines an abstract capability of a Cloud service that can be instantiated in different ways by Cloud service offers, e.g., "authentication". Therefore, abstract features are to be considered by decision-makers in the CSSP. An abstract feature corresponds to a variability point because it potentially contains multiple instance features, one or some of which can be selected to instantiate the abstract feature.

$$p_{e_i} \in V^{ab} \rightarrow c_{e_i} \in V^{in} \forall e_i \in E^{de} \tag{7}$$

The parent feature of an abstract feature must be a grouping feature.

$$c_{e_i} \in V^{ab} \rightarrow p_{e_i} \in V^{gr} \forall e_i \in E^{de} \tag{8}$$

Instance Feature: An instance feature represents a concrete capability of a single Cloud service offer. For example, "OAuth" can be an instance feature for the abstract feature "authentication". Instance features can be decomposed into instance features and represent additional variability points.

$$p_{e_i} \in V^{in} \rightarrow c_{e_i} \in V^{in} \forall e_i \in E^{de} \tag{9}$$

Model Types. CFMs and their configurations represent artifacts within the Cloud service selection process (CSSP) presented in Section 4. We propose to differentiate distinct model types based on the usage of models in the CSSP. Three of the model types are themselves CFMs, namely *domain model*, *service model* and *requirements model*, whereas the fourth type, the *alternative model*, can be represented by other notations. In the following, for each CFM type, its purpose and formal model within the overall CSSP are presented followed by an example.

Domain Model: The domain model's purpose is to represent all relevant abstract decision aspects of the selection problem, e.g., technical, legal or business ones. The domain model does not state the concrete realization of a Cloud service offer or requirements of a decision-maker, but serves as a common blueprint for both service and requirements models. Service and requirements models must not address all abstract decision aspects stated in the domain model. Therefore, the domain model itself does not require to contain variability. Abstract decision aspects are represented by abstract features and hierarchically structured by grouping features. Therefore, a domain model $G_D = (V_D, E_D, FAT_D)$ only includes a reduced set of features.

$$V_D \backslash (V_D^{gr} \cup V_D^{ab}) = \emptyset \tag{10}$$

A domain model contains a set of feature attribute types FAT to specify the non-functional properties to regard in the CSSP. Furthermore, a domain model does not contain cross-tree relationships.

$$E_D^{cr} = \emptyset \tag{11}$$

By acting as a common basis, the domain model supports aggregation of and comparability between service and requirements models. Our approach uses a single domain model within each CSSP. The example domain model illustrated in figure 1 A describes which aspects to regard and how to structure them for selecting a "Cloud Compute Service". For example, the abstract feature "machine size" can contain each service candidate's machine sizes in the corresponding service models. The grouping feature "Security / Legal" categorizes abstract features addressing this concern. Furthermore, attribute types like "cost" with the unit "Euro / hour" define quantitative aspects to regard.

Service Model: A service model $G_{S_D} = (V_{S_D}, E_{S_D}, FAT_{S_D})$ represents a single concrete Cloud service offer considered in the CSSP. A service model's purpose is to capture how this Cloud service offer concretely realizes the abstract decision aspects defined in the domain model. Service models encompass the configurability of Cloud services with regard to these aspects. Therefore, a single service model represents multiple configurations of a Cloud service. A service model G_{S_D} contains the same grouping and abstract features as the domain model G_D it relates to.

$$V_{S_D}^{gr} \cup V_{S_D}^{ab} = V_D^{gr} \cup V_D^{ab} \tag{12}$$

Additionally, it includes instance features to represent the modeled Cloud service's implementation options. Service models also contain the same attribute types as the domain model they relate to.

$$FAT_{S_D} = FAT_D \tag{13}$$

Therefore, each instance feature can be described by feature attributes that relate to one of the attribute types defined in the corresponding domain model. Service models may also contain cross-tree relationships that limit the number of possible configurations. Ultimately, a service model is a CFM without additional design constraints. An example service model is illustrated in figure 1 B. It describes a specific Cloud compute service called "Cloud Service x". It has, for example, two instance features for "machine size", one of which can be chosen, namely "small" or "big". Attributes attached to the instance features denote the instances' "disk space" and "cost".

Requirements Model: A requirements model $G_{R_D} = (V_{R_D}, E_{R_D}, FAT_{R_D})$ represents the requirements that a decision-maker has with regard to the abstract decision aspects in a CSSP. Similar to service models, requirements models contain the same grouping and abstract features as the domain model they relate to.

$$V_{R_D}^{gr} \cup V_{R_D}^{ab} = V_D^{gr} \cup V_D^{ab} \tag{14}$$

Requirements models also contain the same attribute types as the domain model they relate to.

$$FAT_{R_D} = FAT_D \tag{15}$$

This provides the decision-maker with the same structure of decision-relevant aspects as stated in the domain model, thus guiding the requirements stating. The following requirements statements are possible:

- **Any instance feature required:** This requirement states that an abstract feature needs to be instantiated. It does, however, not matter which specific instance feature is chosen if multiple exist. To capture this requirement in the requirements model, the abstract feature is marked *required* using a dedicated property.
- **Specific instance feature required:** This requirement states that an abstract feature needs to be instantiated by a specific instance feature. To model this requirement, the required instance feature is modeled in the requirements model underneath the corresponding abstract feature.

Within the requirements model, a decision-maker can further state requirements regarding values of attributes. The following statements are possible:

- **Threshold for attribute value:** This requirement states that the attribute value of the chosen alternative needs to conform to the specified threshold. To model this requirement, a subset of the corresponding attribute type's domain $dom(v_i.x)$ is chosen in the requirements model to determine the set of valid instantiation values.
- **Specific numerical value required:** This requirement states that the attribute value of the chosen alternative needs to match the specified value. To model this requirement, a specific value for the corresponding attribute type is chosen in the requirements model to determine the valid instantiation value.

The stated requirements represent *minimum* requirements. This means, for example, that if a required instance feature is contained next to other instance features in a service configuration, the configuration fulfills the stated requirements. An example requirements model is illustrated in figure 1 C. The model states that the attribute "cost" needs to be $< 2 Euro/hour$ for a service (configuration) to be considered in the selection. Further, the abstract feature "geographical location" is marked as *required* and instance feature "Europe" is modeled and thus required.

Alternative Model: An alternative model represents a single valid *configuration* derived from a service model, thus a concrete candidate to be selected. Alternative models are not CFMs. Rather, they are represented in a format that allows to use them in further selection steps (i.e. ranking based on preferences). Because alternative models represent valid configurations of a Cloud service, they might be used as basis to deploy the service. Alternative models list the features contained in the configuration they represent. Further, aggregated values for each attribute type defined in the corresponding domain model are contained. An example alternative model is illustrated in figure 1 D. It represents a valid configuration of the "Cloud Service X" service. The presented alternative model fulfills all requirements stated in the requirements model in figure 1 C.

3.3 Model Type Transitions

The transitions between the different model types are illustrated in figure 2. The domain model is the basis for all other models by defining the structure of grouping and abstract features and the relevant attribute types. Service and requirements models are derived from a domain model by adding instance features. Adding optional instance features increases the configurability of the service or requirements model if compared to the domain model: all configurations of the domain model are possible plus additional configurations in which the instance features are contained. Adding optional instance features thus corresponds to the unidirectional refactoring *add optional node* defined by [1]. Adding mandatory instance features, however, does not increase the configurability because is no longer possible to configure the service model in a way that no instance feature is chosen. Thus, overall the transition from a domain model to a service or requirements model can best be described as an *arbitrary edit* [14]. To denote the move from a completely abstract model to a model containing concrete instance features, we refer to this arbitrary edit as *instantiation*. In our approach, the service models define all possible configurations on how to use a Cloud service. Each configuration is potentially represented by an alternative model. We delimit the set of alternative models by determining only those configurations that

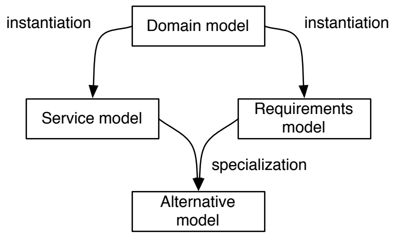


Fig. 2. Transitions between model types

fulfill the constraints stated in the requirements model. Thus, overall the process of deriving alternative models from service models under consideration of the requirements model removes configurations from the service models, which is in literature denoted as *specialization* of a feature model [5].

4 Cloud Service Selection Process (CSSP)

The models and their transitions defined in Section 3 are basis for the CSSP that aims to select Cloud service configurations that fulfill requirements and align best with a decision-maker's objectives. The CSSP involves two roles to execute different tasks and communicate in immediate interactions: Firstly, the role *(human) decision-maker* might be filled by a single or multiple (human) actors. Secondly, the role *decision support tools* presumes software tool implementations of our approach that return Cloud service selection recommendations based on given model inputs. Figure 3 depicts the process and included roles in Business Process Model and Notation (BPMN). The process is triggered by decision-makers that intend to select a Cloud service. Their task is to define multiple models: a domain model, multiple service models and a requirements model. Subsequently, the tool processes the models as input, and generates and evaluates viable Cloud service configurations into a list of recommendations. A detailed description of the activities is given in the following:

- **Define domain model:** Initially, a domain model must be devised to fix a feature hierarchy for future service and requirements models. The modeling process can be self-contained and repetitive. However, eventually, the human decision-makers who participate must release a CFM in a consent to finish the activity.
- **Define service models and requirements model:** Given a complete domain model, next, the current Cloud service landscape can be reflected in service models that follow the domain model's structure. In parallel, in a requirements model, constraints on features and attributes can be set.

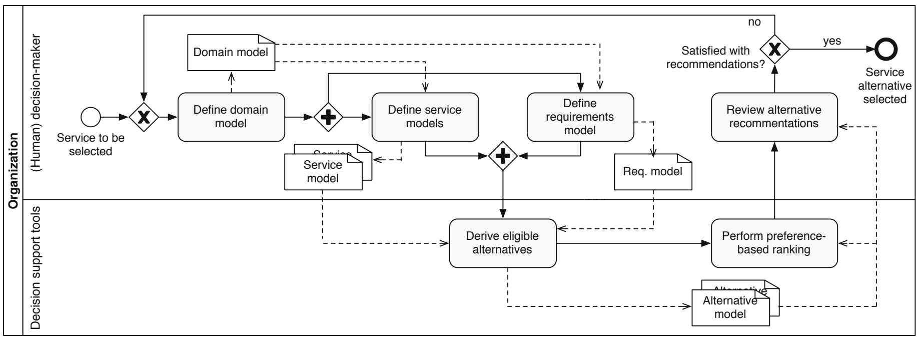

Organization
(Human) decision-maker
Decision support tools
Service to be selected
Domain model
Define domain model
Define service models
Service model
Define requirements model
Req. model
Satisfied with recommendations?
no
yes
Service alternative selected
Review alternative recommentations
Derive eligible alternatives
Perform preference-based ranking
Alternative model

Fig. 3. Cloud Service Selection Process in BPMN

However, some constraints are based on instance features of existing service models (see section 3.2) and, hence, premise a prior modeling of a subset of service models.
- **Derive eligible alternatives:** Based on the set of service models, eligible alternatives can be generated. This can be done by matching the stated requirements of a requirements model with all service models for according node pairs to find viable service configurations, since both models are following the structure of one domain model. Matching can be performed similar to approaches in policies, where name spaces are compared. Alternative models are employed to represent eligible, alternative configurations in a map-like structure.
- **Perform preference-based ranking:** Resulting alternative models can serve as a basis for multi-criteria decision-making (MCDM) methods that allow to obtain a preference-based ranking of the alternatives. Decision criteria can be defined upon the attributes of alternative models derived from service models. Few other Cloud service selection approaches based on MCDM, e.g., the $(MC^2)^2$ framework [10] using an analytic hierarchy process, mention the need for extensive requirements elicitation and filtering. Our approach replaces their alternative definition and simple requirements checks.
- **Retrieve alternative recommendations:** The recommended alternative can subsequently be assessed by a decision-maker. In case of satisfaction, the CSSP stops. Otherwise, further iterations of the process are possible as discussed below.

Our approach allows for cycles to foster evolutionary Cloud service selection. Decision-makers can start with simple models and few requirements and incrementally improve models to strive for more precise results. After reviewing recommendations, a decision-maker can step back and either improve the domain model, service models, or requirements model. However, changes in a model can affect the validity of other models, e.g., an extended domain model requires to change all other models and changing a service model might induce changes in the requirements model. On the other hand, the requirements model allows frequent adjustments to the set of requirements, which is supposed to be common. The process depicted in Figure 3 illustrates the cycle with an optional path from reviewing recommendations back to the start activity (i.e., domain model definition). Entering a new cycle, a decision-maker has the chance to alter models if needed and gain new service recommendations. Dependencies between models have been addressed in Section 3.3 and their management is not automated in our approach yet, and, hence, left to a modeler.

5 Use Case: Cloud Storage Selection

To evaluate the applicability of our modeling approach and subsequent selection process, we applied it to a use case that aims select among *Cloud storage services.* Cloud storage services allow to easily upload data to Cloud data centers, for instance for backup purposes or to share and access data on diverse devices, e.g., mobile phones. The use case is relevant for (technical) decision-makers in search for decision support to select Cloud storage services.

5.1 Implementation

We created a prototypical implementation of a Cloud feature model editor. The implementation builds upon our previous tool suite for service feature modeling [16] and is based on the Eclipse Modeling Framework (EMF)[2]. Our editor provides a graphical interface to model CFMs, including attribute and feature types. We also implemented a reasoning engine that determines the possible configurations for a given CFM and aggregates the attributes based on the standard aggregation strategy defined in the attribute type (see section 3.2). The reasoning engine uses the freely available CHOCO constraint satisfaction problem solver[3], that has also been successfully used by other researchers for such purposes [2]. A requirements matching module determines alternative models in text-form that fulfill the needs stated in the requirements model. The implementation of our tool is publicly available[4].

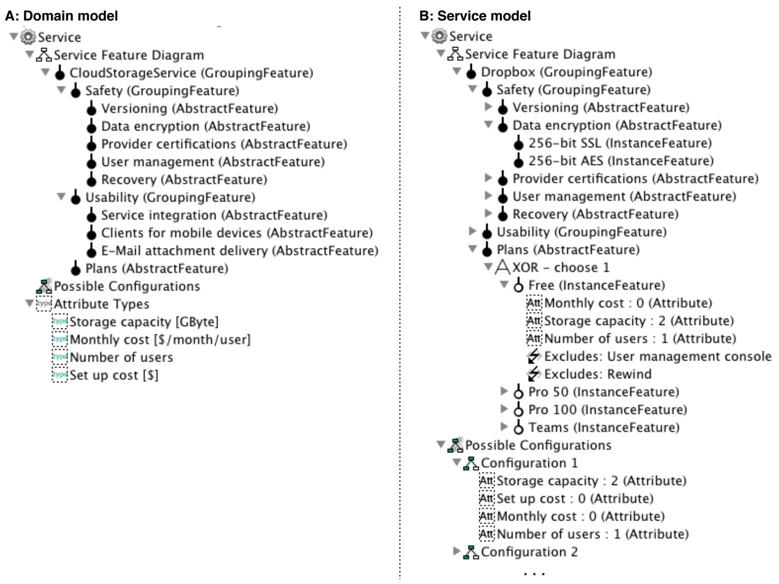
5.2 Performing the Use Case

We performed the use case with regard to those parts of the CSSP that concern modeling and derivation of alternatives, leaving the preference-based ranking open because it is not focus of this work (for details about this step, see for example [10,11,13]).

Definition of Domain Model: We defined a domain model to specify all aspects to be included in the selection. These aspects are derived from published descriptions of the services in focus[5]. As illustrated in Figure 4 A, we defined two grouping features “safety” and “usability”. We used abstract features to denote the aspects whose concrete instantiations we want to assess, e.g., “versioning” and “data encryption”. The abstract feature “plans” relates to offered combinations of “data volume” and “price”. Attribute types denote quantitative characteristics to regard in the selection, for example, “storage capacity” in GByte and “monthly cost” in Dollar.

Definition of Service Models: Based on the domain model, we created 3 service models that represent a random selection of Cloud storage offers: “Dropbox”, “Box.net” and “Wuala”. Per service model, we specified concrete capabilities in form of instance features and attributes based on the publicly available service descriptions. Figure 4 B illustrates parts of the service model we created for Drobpox. Instance features are used, for example, to indicate the “data encryption” mechanisms that Dropbox offers. Attributes denote, for example, the “monthly cost” for the “free” plan, which are 0. Dropbox’s “user management

[2] `http://www.eclipse.org/modeling/emf/`
[3] `http://www.emn.fr/z-info/choco-solver/`
[4] `https://github.com/ErikWittern/CloudServiceSelection`
[5] `http://www.dropbox.com`, `http://www.box.com`, `http://www.wuala.com`



```
A: Domain model
▼ Service
  ▼ Service Feature Diagram
    ▼ CloudStorageService (GroupingFeature)
      ▼ Safety (GroupingFeature)
          Versioning (AbstractFeature)
          Data encryption (AbstractFeature)
          Provider certifications (AbstractFeature)
          User management (AbstractFeature)
          Recovery (AbstractFeature)
      ▼ Usability (GroupingFeature)
          Service integration (AbstractFeature)
          Clients for mobile devices (AbstractFeature)
          E-Mail attachment delivery (AbstractFeature)
        Plans (AbstractFeature)
    Possible Configurations
  ▼ Attribute Types
      Storage capacity [GByte]
      Monthly cost [$/month/user]
      Number of users
      Set up cost [$]
```

```
B: Service model
▼ Service
  ▼ Service Feature Diagram
    ▼ Dropbox (GroupingFeature)
      ▼ Safety (GroupingFeature)
        ▶ Versioning (AbstractFeature)
        ▼ Data encryption (AbstractFeature)
            256-bit SSL (InstanceFeature)
            256-bit AES (InstanceFeature)
        ▶ Provider certifications (AbstractFeature)
        ▶ User management (AbstractFeature)
        ▶ Recovery (AbstractFeature)
      ▶ Usability (GroupingFeature)
      ▼ Plans (AbstractFeature)
        ▼ XOR - choose 1
          ▼ Free (InstanceFeature)
              Att Monthly cost : 0 (Attribute)
              Att Storage capacity : 2 (Attribute)
              Att Number of users : 1 (Attribute)
              Excludes: User management console
              Excludes: Rewind
          ▶ Pro 50 (InstanceFeature)
          ▶ Pro 100 (InstanceFeature)
          ▶ Teams (InstanceFeature)
  ▼ Possible Configurations
    ▼ Configuration 1
        Att Storage capacity : 2 (Attribute)
        Att Set up cost : 0 (Attribute)
        Att Monthly cost : 0 (Attribute)
        Att Number of users : 1 (Attribute)
    ▶ Configuration 2
          . . .
```

Fig. 4. Domain model and service model for "Dropbox" of the use case; screen shots taken from our prototype CFM tool

console" and "rewind" recovery feature are not part of the free plan and thus excluded using cross-tree relationships.

Definition of Requirements Model: Based again on the domain model, we created a simple requirements model. The model states that any "user management" instantiation is required by setting an according property in the abstract feature. Further, attribute type "storage capacity" ought to be $> 60GByte$, while "monthly cost" should be $< 20.00\$/user/month$.

Derivation of Eligible Alternatives: Having the 3 service models and the requirements model in place, we triggered matching of the models to determine those configurations that fulfill the stated requirements. From the original 45 configurations contained in the 3 service models (4 for Box.com, 12 for Dropbox and 29 for Wuala), 5 fulfilled the stated requirements (1 from Box.com and 4 from Wuala).

5.3 Discussion

The use case shows that our modeling approach is capable to efficiently capture the configurations to select from when consuming a Cloud storage service - the 3 service models contained 45 configurations. The requirements elicitation allows to eliminate configurations that do not meet stated requirements, leading in this case to a reduction of configurations by factor 9. The models produced in this use case are publicly available[6]. Because all 3 service models are based on the same domain model, their resulting configurations are comparable to each other. Hence, configurations allow for a service selection already or, after transferring them into adequate alternative models, they can act as input for further MCDM approaches. The application of the use case indicated multiple theses worth following: first, stating the requirements model for existing service models is easy for human decision-makers compared to the creation of the service models, thus allowing to iteratively adapt it if needed. Second, the reuse of existing service models in multiple CSSPs can greatly reduce modeling effort. Third, the collaboration of multiple experts from different domains improves the quality of created service models.

6 Conclusion and Future Work

In this paper, we addressed human-centered modeling of capabilities of Cloud service offers and requirements towards them with a holistic support consisting of models, a process, and a software tool. We adapt feature modeling to capture diverse aspects and configurations of Cloud services: we introduced extended feature attributes, feature types and model types to enable the resulting CFMs to be used within the also presented CSSP. We demonstrate the applicability of our approach in a use case where we compare overall 45 configurations of 3 Cloud storage services. The large number of configurations illustrates how useful it is to utilize variability modeling to represent Cloud services. By applying requirements filtering we showed that we can effectively delimit the number of alternative models, thus generating suitable input for subsequent selection phases (i.e. preference-based ranking).

In future work, we want to involve multiple stakeholders into the modeling of and selection among Cloud services by providing means for collaboration. To better address the dynamics of Cloud computing, we envision to (automatically) incorporate dynamic attribute values into the CFMs that allow for example to consider up to date benchmark results in the Cloud service selection. Next to selecting single Cloud services, the selection and composition of multiple services, concepts for model reuse and an iterative process should be considered.

Acknowledgment. This work was supported by the COCKPIT project [4].

References

1. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., Lucena, C.: Refactoring Product Lines. In: Proc. of the 5th Int. Conf. on Generative Programming and Component Engineering, GPCE 2006, pp. 201–210. ACM, Portland (2006)
2. 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)

[6] `http://bit.ly/CloudServiceSelectionModels`

3. 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)
4. COCKPIT Project: Citizens Collaboration and Co-Creation in Public Service Delivery (2012), `http://www.cockpit-project.eu`
5. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing Cardinality-based Feature Models and their Specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
6. 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)
7. Godse, M., Mulik, S.: An Approach for Selecting Software-as-a-Service (SaaS) Product. In: Proc. of the 2009 IEEE Int. Conf. on Cloud Computing, CLOUD 2009, pp. 155–158. IEEE, Washington, DC (2009)
8. Karataş, A.S., Oğuztüzün, H., Doğru, A.: Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains. In: Bosch, J., Lee, J. (eds.) SPLC 2010. LNCS, vol. 6287, pp. 286–299. Springer, Heidelberg (2010)
9. Li, A., Yang, X., Kandula, S., Zhang, M.: CloudCmp: Comparing Public Cloud Providers. In: Proc. of the 10th Annual Conf. on Internet Measurement, IMC 2010, pp. 1–14. ACM, New York (2010)
10. Menzel, M., Schönherr, M., Tai, S.: (MC2)2: Criteria, Requirements and a Software Prototype for Cloud Infrastructure Decisions. Software: Practice and Experience (2011)
11. ur Rehman, Z., Hussain, F., Hussain, O.: Towards Multi-Criteria Cloud Service Selection. In: Proc. of the 5th Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2011, pp. 44–48. IEEE, Perth (2011)
12. Ruiz-Alvarez, A., Humphrey, M.: An Automated Approach to Cloud Storage Service Selection. In: Proc. of the 2nd Int. Workshop on Scientific Cloud Computing, pp. 39–48. ACM, New York (2011)
13. Saripalli, P., Pingali, G.: MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds. In: Proc. of the 4th Int. Conf. on Cloud Computing, pp. 316–323. IEEE, Washington, DC (2011)
14. Thüm, T., Batory, D., Kastner, C.: Reasoning About Edits to Feature Models. In: Proc. of the 31st Int. Conf. on Software Engineering, ICSE 2009, pp. 254–264. IEEE, Washington, DC (2009)
15. 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)
16. 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)
17. Wohlstadter, E., Tai, S., Mikalsen, T., Rouvellou, I., Devanbu, P.: GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions. In: Proc. of the 26th Int. Conf. on Software Engineering, ICSE 2004, pp. 189–199. IEEE Computer Society, Washington, DC (2004)