# Feature-Based Engineering of Compensations in Web Service Environment

Michael Schäfer[1] and Peter Dolog[2]

[1] L3S Research Center, University of Hannover,
Appelstr. 9a, D-30167 Hannover, Germany
Michael.K.Schaefer@gmx.de

[2] IWIS — Intelligent Web and Information Systems,
Aalborg University, Department of Computer Science,
Selma Lagerloefs Vej 300, DK-9220 Aalborg East, Denmark
dolog@cs.aau.dk

**Abstract.** In this paper, we introduce a product line approach for developing Web services with extended compensation capabilities. We adopt a feature modelling approach in order to describe variable and common compensation properties of Web service variants, as well as service consumer application requirements and constraints regarding compensation. The feature models are being used in order to configure the compensation operations that are applied. In this way, we ensure that the compensation actions are limited to the prescribed ones, and the infrastructure which uses them can be adapted easily in case environment conditions change.

**Keywords:** Software Product Lines, Feature Model, Web Services, Compensations, Business Activities, Transactions.

## 1 Introduction

Web service environments are being used to connect clients and service providers and to establish and maintain conversations between them. Businesses adapt and change their business processes and operations, and they perform transactions with different clients at different times. Their services are accessed by third parties in a concurrent way. Concurrent access to services and changes regarding business processes imply that *service providers* should provide different variants of their services to satisfy the varying needs of different clients and to enable forward recovery for business transactions by replacement with another suitable variant if certain conditions are met. Also, *clients/service consumers* should be able to cover criteria in requirements and constraints assuming that the operations can change and can be replaced by other operations if a failure occurs or certain conditions are met.

We propose a feature based method for engineering compensations in Web service environments. We adopt a method and a modelling technique based on feature models described previously in UML [4]. The infrastructure which utilizes the models is based on our compensation environment described in [10].

The infrastructure uses the XML schema used also within the eclipse plugin for feature oriented domain analysis [1] to provide the technical means for runtime decisions about compensations. The paper provides an evidence on how to a software product line method can be adapted for a novel application area which addresses real complex situations in business to business interactions. It also provides an evidence that the variability descriptions can be utilized by a middleware for the decisions about compensations, where the descriptions specify a client's requirements and constraints regarding compensation handling on the one hand, as well as the offered compensation capabilities of a service provider on the other hand.

The remainder or the paper is structured as follows. A feature modeling based method for web service compensation engineering is discussed in Section 2. Section 3 discusses service provider capabilities conceptual and feature modelling. Section 4 discusses client requirements conceptual and feature modelling, algorithm which ranks providers according to a matching score between capability and compensation model and requirements model, as well as resulting restriction model which serves as a contract between the client and the provider. Section 5 discusses it in the context of related work. Section 6 concludes the paper with a summary and proposal for further work.

## 2 Feature Based Development for Compensations

Software product line methodologies [8] employ a common process pattern. *Domain Engineering* is a process in which the commonality and variability of the product line are defined and realized. *Application Engineering* is a process subsequent to the domain engineering in which the applications of the product line are built by reusing domain artifacts and exploiting the product line's variability.

The domain engineering activities in Web service environments are realized by different independent service providers. The application engineering activities are realized by different parties as well, employing service selection mechanisms and matchmaking to fit particular business activities when utilizing Web services from different providers. Some of the variable features of the Web services can be considered at runtime. Therefore, the software product line engineering process can be tailored to the Web service environment with extended compensation capabilities as follows. Service provider tasks are (capabilities and compensations engineering):

– *Service Domain Analysis* — is a domain engineering process where variabilities and commonalities between service variants are designed to support compensations based on failures or based on different constraints and requirements;
– *Service Domain Design and Implementation* — different service features are mapped onto an implementation and an architecture for service provisioning where some of the features need not to be exposed to the public and some of the variabilities may be left to runtime adaptation.

Client/service consumer tasks are (Requirements and Restrictions):

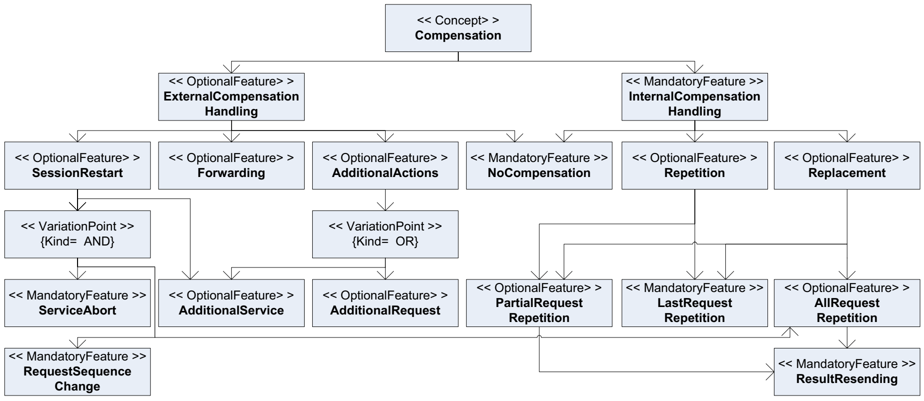
– *Business Application Analysis and Design* — is an application engineering task which may be performed by a party external to the service provider and involves the definition of requirements for and constraints on the Web service compensations;
– *Retrieving the Abstract Web Services* — is an application engineering task in which a designer looks for and retrieves Web services which are required to perform business to business conversations;
– *Defining Client Side Compensations* — is an application engineering task in which a designer defines a variability for compensations which will be exploited at runtime if more Web services with similar capabilities have been found, or an alternative Web service has been defined by an application developer;
– *Implementing Client Side Compensations and Functionalities* — is an application engineering task in which the additional compensations are implemented at the client side, as well as additional operations for which there was no Web service found are realized by an application developer.

As a means for analysis and design we adopt a *feature modelling* approach and a methodology from [3]. *Feature models* are *configuration views* on concepts from conceptual models. The *conceptual model* describes the main concepts of a domain and linguistic relationships between them. *Web service capabilities or client requirements* main concepts are therefore placed into the *application domain conceptual model* and the *compensation concepts* are placed into the *environment conceptual model.* The *functionality feature model* as well as the *compensation feature model* describe the configuration views. Subsequently, the functionality and compensation models are merged to describe the offered capabilities by a service provider, or requested functionalities and restrictions regarding compensations by a service consumer. Different algorithms can then be employed by different middlewares and abstract service to match feature models of a client and service provider, and to trigger forward recovery by utilizing compensation actions agreed on by the consumer and the provider.

## 3 Capabilities and Compensations of Service Providers

*Capability Conceptual and Feature Model.* The capabilities conceptual model describes the concepts from a service application domain and relationships between them. In our case for example, the capabilities conceptual model contain concepts related to payroll processing such as salary, salary transfer, tax, tax rates, employee, and so on. The UML class diagram is used to model such conceptual model.

The capability feature model specifies the capabilities of an abstract service. This model can be provided in the public description of the service and can be used in the client's search process for services that fulfill his requirements. The functionality feature model describes the features of the abstract service that constitute the offered operations that can directly be used in the business process, e.g. the booking of a flight. It can be defined as a normal feature model.



<< Concept> > Compensation
<< OptionalFeature> > ExternalCompensation Handling
<< MandatoryFeature >> InternalCompensation Handling
<< OptionalFeature> > SessionRestart
<< OptionalFeature> > Forwarding
<< OptionalFeature> > AdditionalActions
<< MandatoryFeature >> NoCompensation
<< OptionalFeature> > Repetition
<< OptionalFeature> > Replacement
<< VariationPoint >> {Kind= AND}
<< VariationPoint >> {Kind= OR}
<< MandatoryFeature >> ServiceAbort
<< OptionalFeature> > AdditionalService
<< OptionalFeature> > AdditionalRequest
<< OptionalFeature> > PartialRequest Repetition
<< MandatoryFeature >> LastRequest Repetition
<< OptionalFeature> > AllRequest Repetition
<< MandatoryFeature >> RequestSequence Change
<< MandatoryFeature >> ResultResending

**Fig. 1.** The compensation feature model

*Compensations Conceptual and Feature Model.* In order to describe the available compensation types, a conceptual model is created, which constitutes the basis for the feature models in the extended transaction environment. The result is the *compensation concept model*, usually modeled by a class diagram. The basic concept used in such a model is the *Compensation*, which defines the required compensatory operations for a specific situation in a *CompensationPlan*. Each plan consists of one or more single *CompensationActions*.

The *compensation feature model* describes the configuration aspect of the mandatory and optional features of the compensation concept, and is depicted in Figure 1. It will be used in the next step to define service-specific feature models.

The two main features of this model are the *InternalCompensationHandling* and the *ExternalCompensationHandling* features. They structure the available compensation types as features according to their application: *Repetition* and *Replacement* are only available for internal compensation purposes, while *SessionRestart*, *Forwarding* and *AdditionalActions* are only available for external compensation operations. The exception to this separation is *NoCompensation*, which is the only common compensation feature. Only two of these features are mandatory, the *NoCompensation* and the *InternalCompensationHandling* feature. This is due to the fact that the default compensation action is inactivity: If no rule or compensation capabilities exist, then the service has to fail without any other operations. Accordingly, the ability to perform external compensations is only optional.

The *Repetition* feature contains the subfeatures *LastRequestRepetition* (mandatory) and *PartialRequestRepetition* (optional). *LastRequestRepetition* is mandatory, because even if partial request resending is applied, it will be necessary to resend the last request. Likewise, the *Replacement* feature requires that after the replacement of a concrete service has been performed at least the last request will be resent. Both, the resending of a part of the requests or all requests, requires that it is possible to resend new results to the client. Therefore, the *ResultResending* feature is mandatory.

The *SessionRestart* feature has as an optional subfeature the invocation of an additional service (*AdditionalService*), and requires via an AND variation point the *ServiceAbort*, *RequestSequenceChange*, and *AllRequestRepetition* subfeatures. The capability to abort the service, to change the request log, and to resend all requests is needed in order to perform the session restart, and therefore these three features have to be included. Within an externally triggered compensation, it is possible to invoke additional services and to create and send additional requests to the concrete service. That is why *AdditionalActions* includes the *AdditionalService* and *AdditionalRequest* subfeatures. They are connected via an OR variation point, as the *AdditionalActions* feature needs at least one of these two features.

*Merging Capabilities and Compensations.* The service provider provides at the end only one model to one client. The model is merged from capabilities and compensation feature model. The capability feature model can be extended with a special attribute: A *costs* attribute can be added to each feature. The provider can thus define how much the execution of a specific feature will cost.

## 4 Requirements and Restrictions of Client Application

*Requirements Feature Model.* The client creates a requirement description in order to be able to initiate a search for a suitable abstract service. The specification is being done in the same way as the definition of the capability feature model described in the previous section: A common model is being created that includes the required functionality and compensation features. This model is called the *requirement feature model.* However, although the basic process of creating the requirement feature model is the same, the interpretation of the mandatory/optional properties differs. A mandatory feature *has* to be provided by the service and is thus critical for the comparison process, while an optional feature *can* be provided by the service, and is seen as a bonus in the evaluation of the available services.

*Model Comparison Algorithm.* In the client's search process, each abstract service's capability feature model will be compared to the client's requirement feature model. We define a comparison algorithm which makes it possible to automatically assess the available services and to decide which ones meet the requirements. Our algorithm is a variant of graph matching algorithm on attributed graph [9]. The feature models are attributed graphs where each node is a feature with an attribute stating whether a feature is mandatory or optional. We make use of these attributes in comparing requested capability graph (feature model) with provided capability graph (feature model). The two models are the input for the algorithm, which iteratively compares them and calculates a numerical *compatibility score.* The basic algorithm of comparing the two models functions as follows:

– Using the requirement feature model as a basis, the features are compared stepwise. In this process, it is necessary that the same features are found in the same places, as the same feature structure is expected.
– Each mandatory feature from the requirement model has to be found in the capability feature model as well. A mandatory feature that is found in the capability feature model will not change the compatibility score. If the capability model is missing a mandatory feature, the comparison fails and a negative score is returned to indicate that the service does not fulfill the minimum requirements.
– Each optional feature of the requirement model can be part of the capability model, but does not have to. However, each optional feature that can be found in the capability model counts as a bonus added to the compatibility score. This accounts for the fact that an abstract service that provides more than the absolutely required features is better, as it can more easily be used in different applications and environments.
– Additional features in the abstract service's capability model like the specification of additional services used in the compensation process have to be defined in the correct place, i.e. as a subfeature of the *AdditionalService* feature. Any other additional features will lead to a failure of the comparison.

The compatibility score that is returned by the comparison algorithm describes the degree to which the abstract service fulfills the requirements specified by the client. The requirement model's mandatory features do not increase the compatibility score if they are found in the capability model, because they constitute the minimum requirements. Therefore, an abstract service that provides only the mandatory features has a compatibility score of *0*, although it meets the client's requirements. Each optional feature provided by the service increases the score by a predefined value. The default value for this is *1*, so an abstract service that offers all mandatory features and 5 optional ones has a score of *5*. The higher the compatibility score of an abstract service is, the better it meets the requirements of the client. Using this simple score, it is possible to compare different abstract services and their offered capabilities.

*Restriction Feature Model.* After the client has found and decided upon the necessary abstract services that offer the required functional and compensation features, a contract will be exchanged or negotiated with each service. A vital part of this contract is the specification which compensation features the abstract service is allowed to use for the purpose of processing internal and external compensations. While it is of course possible to apply this restriction by simply searching for abstract services that are able to perform only the allowed compensation actions, such an approach significantly reduces the available services. Moreover, it is quite possible that a client wants to use the same abstract service in multiple applications, each application having its own rules regarding the compensatory actions that are permitted. Therefore, it is beneficial to use a *restriction feature model* that can be part of the contract, and to which the abstract service dynamically adapts its compensation operations.

When the abstract service wants to invoke a specific compensation action, it will first consult the contract's restriction feature model. If the compensation action is part of the model, then the abstract service is allowed to use it. This way, the service can dynamically adapt to the requirements of each single client. It is possible to use an optional attribute in the restriction feature model in order to further restrict the execution of compensatory actions by the abstract service. The client can add a *maxCosts* attribute to the *InternalCompensationHandling* and *ExternalCompensationHandling* features, which specifies the maximum costs that may be spent by the abstract service for internal and external compensation handling, respectively. Using this approach, it is possible to define a "budget" for internal or external compensation handling.

*Feature Model Specification for Middleware.* The *FeaturePlugin* [1] for Eclipse has been applied to create the compensation feature model to be able to obtain an XML version of the feature models to be used by our transactional environment. This feature model is used as a basis for the specification of capability and requirement feature models, by changing the mandatory/optional features, or by deleting parts of the model. A restriction feature model can be created in a convenient way as a configuration of the predefined compensation feature model. While doing so, the plug-in monitors the constraints and thus guarantees that the resulting restriction feature model is valid with respect to the properties of the features as well as the feature group cardinalities.

## 5 Related Work

A product line approach for composite service-oriented systems has been envisioned in [2]. Our approach contributed to the issues of service selection, exception handling and quality factors identified in that paper in the context of service compensations. A pattern based variability has been employed for development of composite service-oriented systems in [7]. Our approach is based on feature models for variability description. The compensation mechanisms, engineering methodology and infrastructure can be used as a supplement to the method presented in [7]. [5] studies product lines in the context of adaptive composite service oriented systems. Our approach can be used as a supplement to provide compensations in such environment to support forward error recovery. [11] defines an atomicity-equivalent process algebra to define public views over business processes involved in B2B conversations. Views are used to check whether the processes are still in an atomicity sphere; i.e. the process is guaranteed to terminate with semantics all or nothing. Our approach allows for other semantics to satisfy clients at least partially though we need to study the properties of termination further. [6] deals with graph matching for feature composition from partial feature models as well. In our approach we do not compose feature models, we try to find out which service fits the client requested capability.

## 6 Conclusion and Future Work

We have described a software product line approach to be used for Web service transactions in order to control the use of compensatory actions. The compensation feature model has been introduced that structures the compensation types and activities. This model has subsequently been used in order to define the feature models for service capabilities, requirements, and restrictions.

It is necessary to run additional experiments with different scenarios, and to further analyze the usability of feature models. It is interesting to study the use of ontologies in the models. The extensions of the model comparison algorithm should also be studied.

## References

1. Antkiewicz, M., Czarnecki, K.: Featureplugin: Feature modeling plug-in for eclipse. In: OOPSLA 2004 Eclipse Technology eXchange (ETX) Workshop (2004)
2. Capilla, R., Topaloglu, N.Y.: Product lines for supporting the composition and evolution of service oriented applications. In: Eighth Intl. Workshop on Principles of Software Evolution in conjunction with ESEC/FSE 2005 (2005)
3. Dolog, P., Nejdl, W.: Using UML-based feature models and UML collaboration diagrams to information modelling for web-based applications. In: Baar, T., Strohmeier, A., Moreira, A., Mellor, S.J. (eds.) UML 2004. LNCS, vol. 3273, pp. 425–439. Springer, Heidelberg (2004)
4. Dolog, P.: Engineering Adaptive Web Applications: A Domain Engineering Framework. VDM Verlag Dr. Müller (2008), `http://www.vdm-publishing.com/`
5. Hallstein, S., Stav, E., Solberg, A., Floch, J.: Using product line techniques to build adaptive systems. In: SPLC 2006. 10th Intl. Software Product Line Conf. (2006)
6. Jayaraman, P.K., Whittle, J., Elkhodary, A.M., Gomaa, H.: Model composition in product lines and feature interaction detection using critical pair analysis. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 151–165. Springer, Heidelberg (2007)
7. Jiang, J., Ruokonen, A., Systä, T.: Pattern-based variability management in web service development. In: ECOWS 2005. Third European Conf. on Web Services (2005)
8. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering. Springer, Heidelberg (2000)
9. Rozenberg, G.: A Handbook of Graph Grammars and Computing by Graph Transformation: Application Languages and Tools. World Scientific Publishing Company, Singapore (1997)
10. Schäfer, M., Dolog, P., Nejdl, W.: Environment for flexible advanced compensations of web service transactions. ACM Transactions on Web 2(2) (April 2008)
11. Ye, C., Cheung, S.C., Chan., W.K.: Publishing and composition of atomicity-equivalent services for b2b collaboration. In: ICSE 2006: Proceedings of the 28th Intl. Conf. on Software Engineering. ACM, New York (2006)