Toward a QoS Based Run-time Reconfiguration in Service-oriented Dynamic Software Product Lines

Jackson Raniel Florencio da Silva, Aloisio Soares de Melo Filho and Vinicius Cardoso Garcia
Center of Informatics, Federal University of Pernambuco, Recife, Pernambuco, Brazil

Keywords: SPL, DSPL, SOA, Software Product Line, Dynamic Software Product Line.

Abstract: Ford invented the product line that makes possible to mass produce by reducing the delivery time and production costs. Regarding the software industry, this, roughly presents both a manufacturing and mass production that generates products that are denoted as individual software and standard software (Pohl et al., 2005): a clear influence of Fordism in the development paradigm of Software Product Lines (SPL). However, this development paradigm was not designed to support user requirements changes at run-time. Faced with this problem, the academy has developed and proposed the Dynamic Software Product Lines (DSPL) (Hallsteinsen et al., 2008) paradigm. Considering this scenario, we objective contribute to DSPL field presenting a new way of thinking which DSPL features should be connected at run-time to a product based on an analysis of quality attributes in service levels specified by the user. In order to validate the proposed approach we tested it on a context-aware DSPL. At the end of the exploratory validation we can observe the effectiveness of the proposed approach in the DSPL which it was applied. However, it is necessary to perform another studies in order to achieve statistical evidences of this effectiveness.

1 INTRODUCTION

Cost, reusability and ”time-to-marketing” are aspects of productivity that concern software manufacturers. Moreover, the software configuration management becomes increasingly more complex as they increase the possible combinations of software features. The paradigm of Software Product Lines (SPL) address this problem by creating software products from a common set of characteristics. The management of the configuration occurs with the variabilities and commonalities between members of the SPL.

According to the Software Engineering Institute (SEI)[1], a SPL is a set of software-intensive systems that share a common, managed set of features, satisfying the specific needs of a particular market segment or mission. The features are developed from a common set of core assets in a prescribed way. The benefits of adopting the SPL paradigm depend on the context in which it is applied. Some of the most common benefits are the reduction of cost and “time-to-marketing” promoted by reusing the SPL core assets.

However, dynamic changes in user requirements and the environment in which the software is housed at run-time are becoming increasingly frequent, so there is an increasing demand for systems that can automatically adapt. In 2008, Hallsteinsen (Hallsteinsen et al., 2008) published a study that described the concept of Dynamic Software Product Lines (DSPL). These product lines differ from others by not managing the variability during the design phase of the software. Instead, it is managed, later at run-time.

The DSPL have five principal characteristics: a) dynamic variability, b) biding of the elements of DSPL changes during the life cycle of the application, c) variation points change at runtime, d) unexpected changes in the context, and e) development for one specific context or environment in place of a market segment (Hallsteinsen et al., 2008).

Optionally, DSPLs may be aware to the context around them (Parra et al., 2009; Ali et al., 2009; Alferez and Pelechano, 2011), have autonomic properties or have self-adaptation (Abbas et al., 2011; Cetina et al., 2008), and have automatic decision making. These characteristics are a challenge to a development model of SPLs that manages the variability out of the design phase.

We propose in this study an approach for analyze which DSPL features should be connected at run-time to a product based on an analysis of quality attributes in service levels specified by the user. The Section

[1]http://www.sei.cmu.edu/productlines/tools/framework/

Florencio da Silva J., Soares de Melo Filho A. and Cardoso Garcia V..
Toward a QoS Based Run-time Reconfiguration in Service-oriented Dynamic Software Product Lines.
DOI: 10.5220/0004973404600465
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 460-465
ISBN: 978-989-758-028-4

2 presents in details the problem addressed, Section 3 presents an overview of the literature about DSPL, the Section 4 presents the proposed approach that are evaluated in Section 5.

2 PROBLEM STATEMENT

This study is under the MobiLine[2] project, which is a DSPL for context-aware mobile applications divided into two abstraction levels: base and specific levels. The base level has common features to the mobile context-aware domain applications and the specific level that consists of features that belongs to a particular business domain (Marinho et al., 2012).

Their applications use contextual information, such as location, profile, visitor user preferences, and requests to adapt their own behavior at run time. The adaptation occurs while connecting features which are Web services to the product on the mobile device.

The MobiLine project objectives the creation of a mechanism for automatic adaptation at run-time, choosing features of the application that will be enabled for the user and adapting them according to the context in which the application operates. The operation of this mechanism may vary based on rules of action or condition, may even be modeled by an optimization problem.

The study by Martins (de Oliveira Martins, 2013) talks about the mechanism of reconfiguration and service discovery framework called DYMOS. In this approach, whenever a reconfiguration occurs on the client side, a reconfiguration happens on the server side. However, this study selects what services are on the server side in an arbitrary manner without making any analysis of services that will compose the DSPL.

Figure 1 represents the hypothetical scenario that exemplifies the problems addressed in this study. As previously reported, the MobiLine product, in run-time execution, collects contextual information (collected by sensors or device status) and receives stimulus from the user with the information which define run time adaptations must occur.

When a mobile device adaptation happens, two situations can occur: a) the request to a Web service present and active on the server side or b) the request of a missing or inactive Web service on the server side.

For situation a), when the service of MobiLine products are found they are consumed as expected. In situation b), after the creation of DYMOS when the service is not found, the framework is responsible

[2]http://mobiline.great.ufc.br/index.php

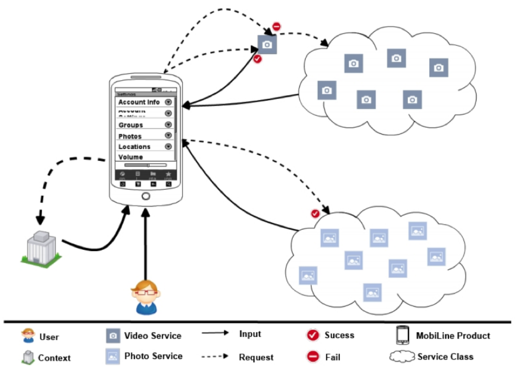

Figure 1: Problem Scenario.

for finding a service to replace what is available, in accordance with an order of priority. If none of the services organized by priority are available, any service that meets the interface required by the mobile application, is now available.

However, Alferez (Alferez and Pelechano, 2011) argue that Web services are implemented in complex and heterogeneous environments, and it is appropriate adaptation mechanisms according to contextual changes affecting reconfiguration and increasing quality of the service provided. According to Lin (Lin et al., 2010), from a business perspective, it is strongly desirable to maximize the quality of a service provided given different understandings of what quality is from different customers' perspectives.

Hence, the aim of this article is to propose an approach for selection of features that will compose a DSPL at run time based on an analysis of the respective quality attributes.

3 LITERATURE REVIEW AND RELATED WORK

The aspects related to the reconfiguration in real time in SPL is a study extensively explored by researchers, which is treated as part of the derivation of the DSPL called dynamic derivation (Parra et al., 2009).

In a systematic way of developing dynamically configurable core assets in a feature oriented approach Lee (Lee, 2006) developed a reconfigurator that accumulates functions to monitor and manage the configuration of a product at run time. The authors believe that the dynamic derivation can be accomplished by grouping features in bind units and using a reconfigurator that considers contexts (when to configure), reconfiguration strategies (how to reconfigure) and reconfiguration actions (what to reconfigure). Latter, in a continuation of the study carried out by the same author (Lee and Kotonya, 2010) a possible solution for dynamic reconfiguration was described that combines a feature-oriented analysis and a framework sensible to the quality of services of an SPL.

Gomaa (Gomaa and Hashimoto, 2011) describes an architecture for dynamic adaptation of SPL products based in Service-Oriented Architecture (SOA). This architecture contains a monitoring service that continuously checks the status of the running system and forwards it to the calibration service that, perceiving a situation in the context that requires an action, sends a request of feature selections to a dynamic device. This device determines if there is a need for configuration changes in the running configuration. It dynamically determines necessary changes for the feature model of the application and sends the new set of features for the change management service. It is in charge of change management service and determines the difference between the new and actual features selection, determines the components and connectors need to be added or removed and generate a sequence of commands that correspond to adapting to the changes that need to be made.

Alferez (Alferez and Pelechano, 2011) developed a method to design and implement autonomous and context-aware Web services in SPL. The authors argue that Web services are characterized by features, and that then the activation and deactivation of features at run-time can guide the autonomic reconfiguration of services composition according to changes in context. In this case, the products are a composition of services. To make it possible for a service composition, the authors suggest the creation of a composition model made from a UML activity diagram. This model illustrates the Web services and the flow sequence between them that determines the order in which each Web service must be activated.

The relationship between SPL and service-oriented computing has clearly used by Yu (Yu et al., 2010) by presenting a methodology for building-based services for heterogeneous and dynamic environments applications. The methodology is divided in phases of domain engineering, application engineering and run-time. A development process focused on domains and inspired by SPL-oriented architecture is proposed. It is during run-time that, through dynamic service composition, the reconfiguration happens.

It is interesting to note that SOA follows an approach of reuse and composition of services whereas SPL corresponds to an approach by building and decomposition (Parra et al., 2009). However, the characteristics of these opposing paradigms (composition and decomposition) are not conflicting since service composition in a traditional SPL comes at a pre-derivation time, and decomposition occurs at the the product derivation moment.

Traditional DSPLs may also not present difficulties for this antagonism. Nevertheless, this problem certainly affects service-oriented DSPLs thats needs to decides based on some analysis (e.g., priority, response to context, and QoS), which services should be plugged into at run-time, because derivation (decomposition) and composition are occurring at the same time.

About this question, we have examples as treated by Yu (Yu et al., 2010) in which the service-oriented DSPL makes the analysis of which services compose the product only checking the availability of services. Additionally, Lee (Lee, 2006) presents a service-oriented DSPL where the features are mapped to services. This DSPL performs an analysis and planning at run time based on contextual changes to define which features and, consequently, what services should compose the product.

In a later study Lee (Lee and Kotonya, 2010) modified their approach. In the modified approach, the run time analysis is made based on service level agreements imposed by product line. In the same manner as in the studies of Alferez (Alferez and Pelechano, 2011) and Gomaa (Gomaa and Hashimoto, 2011), where when a break occurs in the service level agreement the framework looks for another service that satisfies the desired conditions. No details were expressed in the study about what those conditions (QoS attributes) or how the trading is done.

The definition of the services that will compose the service-oriented DSPL based on an analysis of QoS attributes held for each user request is not present in the literature studies. This is the objective of the present study.

4 DYMOS QoS APPROACH

To achieve the objective of this study, we proposed extending the DYMOS (de Oliveira Martins, 2013) approach, which is starting to be called DYMOS QoS. This proposed solution consists of an application developed in JAVA language and Open Services Gateway Initiative (OSGi) and that assesses individual and collectively attributes of service quality available for the reconfiguration of the SPL at run time.

Using DYMOS QoS, it is possible to change the situation b) presented in Figure 1. In this situation, the service selection to replace the unavailable service occur according to the priority arbitrated. With our proposed approach, the services selection occur according to the quality of the services available upon request. It selects the service that has the highest score provided by the framework. Although this framework, makes possible to occur a third scenario where the request for a service ever return a great service for a certain level of service required.

The framework is structured as shown in the Figure 2. This architecture presents three descriptor elements: the *ServiceDescriptor* that describes what can be binded at run-time in the SPL product; the *VariabilityDescriptor* that describes the variation points; and finally the *WSDescriptor* that describes the WSDL of each Web service.

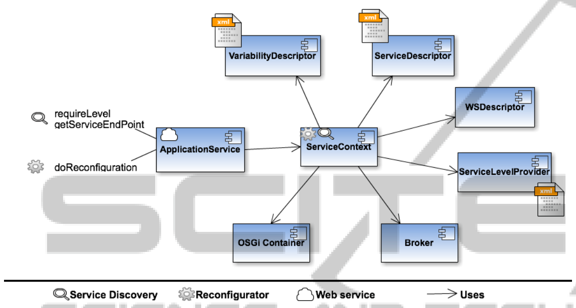

Figure 2: DYMOS QoS Package Diagram.

The *ServiceDescriptor* uses an XML file to describe the services. This XML contains a list of structured fields for each Web service such as a "Id" field to uniquely identify a Web service, a "serviceSpec" and "serviceImpl" fields that contains the values that identify the interface, and an implementation for the service. Each service description has, optionally, a list of alternative services that contains a reference for another Web service.

The variability meta data structure utilized by the *VariabilityDescriptor* defines a list of variabilities with an identification (ID) by the "id" attribute, a name via the "name" attribute, and a set of variants. A variant consists of an ID, a name and a set of service references.

The *WSDescriptor* is used for service discovery operations, and its main function, according to a particular service provided, is describe attributes that are not described in the *ServiceDescriptor* and are specific to each implementation of a service as "ServiceName", "PortType", or "Target Namespace". Thus, *WSDescriptor* allows greater flexibility to the framework, adding dynamic and loosely coupled characteristics.

The components responsible for the QoS analysis are the *ServiceLevelProvider* and the *Broker*. The first one stores contextual information about the Web services stored in a structured way in an XML file and update this information according to changes in the context. The *Broker* component is in charge of analyzing the quality attributes of each service in run time during the service discovery.

In order to be possible to the *Broker* perform that analysis the *ServiceLevelProvider* XML stores some quality attributes, such as service level, maximum capacity, current capacity, response time, and cost. The broker analysis calculates the utility (Yu and Lin, 2005) of each service at the chosen service level and sends to the *ServiceContex* component.

The utility function can be seen in the Equation 1, where w_b and w_c are the weights of the benefit an the cost, $b(s,l)$ and $c(s,l)$ are the benefit and the cost by choosing service s in the level l, avg_b and avg_c are the average benefit and the average cost for the available services in the chosen service level, and finally std_b and std_c are the standard deviation of benefit and cost.

$$F(s,l) = w_b * \left(\frac{b(s,l) - avg_b}{std_b}\right) + w_c * \left(1 - \frac{c(s,l) - avg_c}{std_c}\right) \tag{1}$$

This equation uses the results of the benefit function that are calculated by subtracting the current capacity of a service from their max capacity on a service level and dividing the result by the max capacity in the chosen service level. That means that the benefit is a rate of the Web service load.

The *ServiceContext* component makes use of all descriptors components. The use of these components allows the *ServiceContext* to get all the information described in the form of Java objects, allowing management of variability and services described. This management is to use the information on Variability and services for manipulating the OSGi container.

The component *ApplicationService* was implemented to act as a facade, creating an isolation between the client and the other framework components. Thus, it reduced the number of objects that clients needed to deal with. The main objective of the component is to expose operations, thus allowing the service discovery and the variability management.

These operations are provided by means of Web services, so that all incoming requests should be delegated to the *ServiceContext* component so that they can be treated. This component was implemented using iPOJO to get it in the form of OSGi bundle and DOSGi CXF to expose their functionality through Web services.

5 PRELIMINARY VALIDATION CASE

In order to validate the proposed approach, we then applies it in the products of the MobiLine. To do so, first it was necessary adequate the client side of the DSPL and before configure the DYMOS QoS by making the XML files necessary for the descriptors and *ServiceLevelProvider* components.

To the DYMOS QoS approach to be fully functional, each time a service will be called in the client side the wanted service level and the end point of this service need to be refreshed. This is done by making a call to the requireLevel() and getServiceEndPoint() methods of the *ApplicationService* Web service from the framework.

The variabilities XML handled by the *VariabilityDescriptor* maps the variants to their respective variabilities. Thus, to the framework a variant can be activated by the client just when the variability is activated. This is part of the static derivation process of the MobiLine and is well described by Martins (de Oliveira Martins, 2013).

Listing 1: Services XML File.

```
<?xml version="1.0"?>
<services>
...
<service id="imageService"
service-impl="com.assertLab.imageServiceImpl">
 <service-spec>
  com.assertLab.imageService
 </service-spec>
  <alternative-service ref="imageService1" />
  <alternative-service ref="imageService2" />
  <alternative-service ref="imageService3" />
  <alternative-service ref="imageService4" />
  <alternative-service ref="imageService5" />
</service>
<service id="imageService1"
service-impl="com.assertLab.imageService1">
 <service-spec>
  com.assertLab.imageService
 </service-spec>
</service>
<service id="imageService2"
service-impl="com.assertLab.imageService2">
 <service-spec>
  com.assertLab.imageService
 </service-spec>
</service>
<service id="imageService3"
service-impl="com.assertLab.imageService3">
 <service-spec>
  com.assertLab.imageService
 </service-spec>
</service>
<service id="imageService4"
service-impl="com.assertLab.imageService4">
 <service-spec>
  com.assertLab.imageService
 </service-spec>
</service>
<service id="imageService5"
service-impl="com.assertLab.imageService5">
 <service-spec>
  com.assertLab.imageService
 </service-spec>
</service>
...
</services>
</variabilities>
```

Listing 2: QoS Attributes XML File.

```
...
<serviceLevel id="1">
 <cost>2,07</cost>
 <curCapacity>0</curCapacity>
 <level>1</level>
 <maxCapacity>42</maxCapacity>
 <name>imageService1</name>
 <responseTime>23379</responseTime>
 <serviceSpec>
  com.assertLab.imageService
 </serviceSpec>
</serviceLevel>
<serviceLevel id="2">
 <cost>0,29</cost>
 <curCapacity>38</curCapacity>
 <level>2</level>
 <maxCapacity>48</maxCapacity>
 <name>imageService1</name>
 <responseTime>44639</responseTime>
 <serviceSpec>
  com.assertLab.imageService
 </serviceSpec>
</serviceLevel>
...
```

Each variant in the DYMOS QoS framework can be implemented by a class of Web services that are bound dynamically in run-time to the DSPL. Each Web service in the class implements the same service interface. The services XML, shown in Listing 1, is responsible to describing the services interfaces and implementations. Also, each service implementation has a set of QoS attributes, as shown in Listing 2.

To exemplify the operation of the DYMOS QoS in a real situation we activated the variant "imageService" and all their alternative services described in the Listing 1 with the cost, max capacity (cMax), current capacity (cCurr), and utility function value (UF) shown in the Table 1. We experimented with the three situations described in Section 2.

Table 1: QoS Values.

Service	cMax	cCurr	Cost	UF
imageService1	42	0	2,07	0,82
imageService2	39	9	6,86	-0,58
imageService3	30	17	0,56	0,18
imageService4	23	22	6,97	-1,83
imageService5	43	26	9,69	-1,82

In the first situation the a request for the end point of the variant “imageService” is made in the Mobi-Line Product. The *ServiceContext* uses the descriptor components of the framework to identify the implementation and the specification of the variant. Thus, the service context verifies the availability of the service in the OSGi container and sends the end point to the mobile device.

For testing the second situation, we turned the service “imageService” inactive in the OSGi Container and maintained all the alternative services activated. When a request for the end point of this variant is made the *ServiceContext* requests from the Broker a list of alternative services and their respective punctuation for the utility function. Thus, the *ServiceContext* will return the end point for the required variation by considering the service active with the highest score in the utility function in the OSGi container.

6 CONCLUSIONS

This paper presents an approach to performing the dynamic derivation in service-oriented DSPLs based on a run time QoS analysis. The approach was described and evaluated in a context-aware service oriented DSPL called MobiLine.

We consider, according to our evaluation, that the approach is functional and addresses the problem specified in Section 2. Moreover, interesting next steps are the evaluation of the proposed approach behavior with a large number of services and variabilities. It would be interesting for future investigations too verify the performance of the proposed approach and empirically compares the results with another approaches or with others DSPLs.

ACKNOWLEDGEMENTS

This research is sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq) grants in the process 381305/2011-2. It was also partially supported by the National Institute of Science and Technology for Software Engineering (INES[3]) , funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08. It is also part of UbiStructure - An infrastructure for developing and executing ubiquitous applications - CNPq 14/2011, process number 481417/2011-7.

[3]www.ines.org.br

REFERENCES

Abbas, N., Andersson, J., and Weyns, D. (2011). Knowledge evolution in autonomic software product lines. page 1, New York, New York, USA. ACM Press.

Alferez, G. H. and Pelechano, V. (2011). Context-Aware Autonomous Web Services in Software Product Lines. pages 100–109. Ieee.

Ali, R., Chitchyan, R., and Giorgini, P. (2009). Context for goal-level product line derivation.

Cetina, C., Fons, J., and Pelechano, V. (2008). Applying Software Product Lines to Build Autonomic Pervasive Systems. Number ii, pages 117–126. Ieee.

de Oliveira Martins, D. A. (2013). DYMOS: Uma abordagem para suporte a variabilidades dinâmicas em Linhas de Produto de Software Orientado a Serviços e Sensível ao Contexto. Master’s thesis, Universidade Federal de Pernambuco, Recife, Pernambuco, Brazil.

Gomaa, H. and Hashimoto, K. (2011). Dynamic software adaptation for service-oriented product lines. page 1, New York, New York, USA. ACM Press.

Hallsteinsen, S., Hinchey, M., Park, S., and Schmid, K. (2008). Dynamic software product lines. *Computer*, (April):93–95.

Lee, J. (2006). A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. pages 131–140. Ieee.

Lee, J. and Kotonya, G. (2010). Combining service-orientation with product line engineering. Number June, pages 35–41.

Lin, K.-J., Zhang, J., Zhai, Y., and Xu, B. (2010). The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA. *Service Oriented Computing and Applications*, 4(3):157–168.

Marinho, F. G., Andrade, R. M., Werner, C., Viana, W., Maia, M. E., Rocha, L. S., Teixeira, E., Filho, J. a. B. F., Dantas, V. L., Lima, F., and Aguiar, S. (2012). MobiLine: A Nested Software Product Line for the domain of mobile and context-aware applications. *Science of Computer Programming*.

Parra, C., Blanc, X., and Duchien, L. (2009). Context awareness for dynamic service-oriented product lines. pages 131–140.

Pohl, K., Böckle, G., and van der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*.

Yu, J., Lalanda, P., and Bourret, P. (2010). An Approach for Dynamically Building and Managing Service-Based Applications. pages 51–58. Ieee.

Yu, T. and Lin, K.-J. (2005). Service selection algorithms for Web services with end-to-end QoS constraints. *Information Systems and e-Business Management*, 3(2):103–126.