An architectural discussion on DSPL*

Carlos Cetina, Vicente Pelechano
Dpto. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, E-46022, Spain
{ccetina, pele} at dsic dot upv.es

Pablo Trinidad, Antonio Ruiz-Cortés
Dpto. Lenguajes y Sistemas Informticos
Universidad de Sevilla, Spain
{ptrinidad, aruiz} at us dot es

Abstract

Dynamic Software Product Line (DSPL) engineering has proved itself as an efficient way to deal with run-time product adaptation. DSPLs have been successfully applied in domains such as smart homes, mobile devices or multimedia systems. However, existing DSPLs focus their efforts on highly adaptive products or on autonomic products. In this paper, we classify DSPLs according to their adaptation mechanisms and we also propose mixed DSPL which simultaneously address both adaptivity and autonomy. Finally, we discussed the underlying infrastructure to develop mixed DSPLs.

1 Introduction

A Software Product Line (SPL) engineering pursues the objective of producing a set of products that share a common set of assets in an specific domain. SPL engineering techniques allows to adapt a product to the customer needs decreasing its production costs and time to market. A product adaptation is performed during the SPL development, however further adaptations of a delivered product is hard to be managed.

Increasingly, software needs to dynamically adapt its behavior at run-time in response to changing conditions in the supporting computing infrastructure and in the surrounding physical environment [8]. Adaptive software is able to adapt its properties and resource requirements at run-time in response to dynamically varying user needs and resource constraints.

A Dynamic Software Product Line (DSPL) [4] is a SPL whose products are adaptative systems, i.e. a product might pro actively adapt itself when changes are performed in its environment.

When a product is produced in a SPL, a feature may be bound at different times: design time, compilation time, configuration time, runtime... Intensively using runtime binding, we would be able to produce reconfigurable products that may change their functionality even when they are deployed. However, the main difference between DSPL and SPL is the ability of changing the functionality of a product automatedly and without the interaction of users. This grade of autonomy implies the usage of techniques that provide the knowledge or reasoning ability to adapt a product at runtime.

With the emergence of pervasive, mobile and service oriented computing, dynamics variation in users needs and available resources is becoming more and more widespread. For example, a pervasive system such as an smart home is highly dynamic since new kinds of entities (sensors, actuators, external software systems) can be installed in the system at any time. Furthermore, existing entities may fail or leave the system for a variety of reasons: hardware faults, OS errors, software bugs, network faults, etc. The dynamic of these systems makes SPL address the production of adaptive products.

In this paper we intend to summarize the DSPL architectures that have been proposed at date, dividing them in connected and disconnected DSPL depending on their connectivity and dependence with the DSPL development. We propose mixed DSPL as an intermediate solution that takes the benefits of both connected and disconnected DSPL architectures, giving some details about its internals.

This paper is structured as follows: In Section 2, the differences between SPL and DSPL development are remarked. Section 3 reviews the state of the art on DSPL architectures analyzing them using different aspects. In Section 4, we propose mixed DSPL as an intermediate alternative between connected and disconnected DSPL, whose architectural internals are discussed. Lastly, in Section 5, some conclusions and future work are briefly commented.

*This work was partially supported by the European Commission (FEDER) and Spanish Government under Web-Factories (TIN2006-00472) and SESAMO (TIN2007-62894) projects and by the Andalusian Government under project ISABEL (TIC-2533).

2 Context and Scenarios

SPL main objective is producing products while costs and time-to-market are reduced by an intensive reuse of commonalities and a suitable variability management. Products are commonly produced by selecting the features that are part of a product. To make this decision, features are selected and/or discarded at different binding times. Those features thought to be bound at runtime are kept in the final product even when they may not be used by the final product. The product must provide the mechanisms to select the suitable feature at runtime and optionally reconfigure the product. After the production, no automated activity is specified in SPL development to maintain a product in connection with the SPL so it may benefit from feature updates.

On the other hand, DSPL development (see Figure 2) mainly intends to produce configurable products [1] whose autonomy allows them to reconfigure themselves and benefit from a constant updating. In a DSPL, a configurable product(CP) is produced from a product line similarly to standard SPL. However, the reconfiguration ability implies the usage of two artifacts to control it: the *decision maker* and the *reconfigurator*. The decision maker is in charge of capturing all the information in its environment that suggests a change such as external sensors information or even a user. The analyser must know the whole structure of a CP so it makes a decision on which features must be activated and deactivated. The reconfigurator is responsible of executing the decision by using the standard SPL runtime binding. A CP may be considered as an extension to traditional SPL products where there are no bound features but the decision maker and the reconfigurator and the remaining features are bound at runtime. As a consequence, new features may be added to an existing product or even existing features may be updated at runtime.

Comparing both, SPL and DSPL development, DSPL development might be considered as a particular case of SPL where following properties are added:

- Adaptation to changing requirements and environment: a product is prepared to response to the so called *adaptation triggers* which alert the system to a change in the environment or conditions the product is working in. Besides, a user may explicitly ask for a change in a product configuration. Both cases are analysed for their consequences and if it is possible, the system is reconfigured to adapt itself to new situations.

- Autonomic capabilities: a CP is able to make decisions about the features that are activated and deactivated at a time from the information obtained from its environment and whenever an adaptation trigger or an user request arrives to the decision maker.

- Product Updates: as all the features are bound at runtime, updating an existing product with new features or updates is eased

Analyzing the reasons why a CP is reconfigured in the above context, there are several suitable reconfiguration scenarios [2]. However, we have mainly distinguish two kinds of scenarios:

- Involution Scenario: the new configuration of a CP is performed with the available features, activating or deactivating them. This scenario happens whenever an adaptation is triggered from the system.

- Evolution Scenario: the new configuration of a CP implies the usage of a new feature or the update of an existing one.

The number of involution scenarios is finite, as the potential subproducts of a CP is finite. However, evolution scenarios are difficult to be previewed from a CP as new features unknown to the CP may appear.

3 A DSPL Taxonomy

Several approaches for developing CP using DSPLs have been presented along the published literature. In this work, we have classified these approaches in two categories, according to the way in which product adaptation is considered. These categories are the following:

- **Connected DSPL.** The DSPL is in charge of the product adaptation.

- **Disconnected DSPL.** The CP itself is in charge of the product adaptation.

We describe both connected and disconnected DSPLs from the *SPL perspective* and the *product perspective*. In the SPL perspective, we study the technical extensions incorporated in a SPL to become a DSPL and achieve product adaptation. This technical side is described using the following criteria:

- **Adaptation mechanism.** This criteria addresses how product adaptation is achieve.

- **Contact between the SPL and the product.** This criteria addresses information interchange between the SPL and the product.

- **Variability manager.** This criteria addresses which SPL participants need capabilities to manage variability.

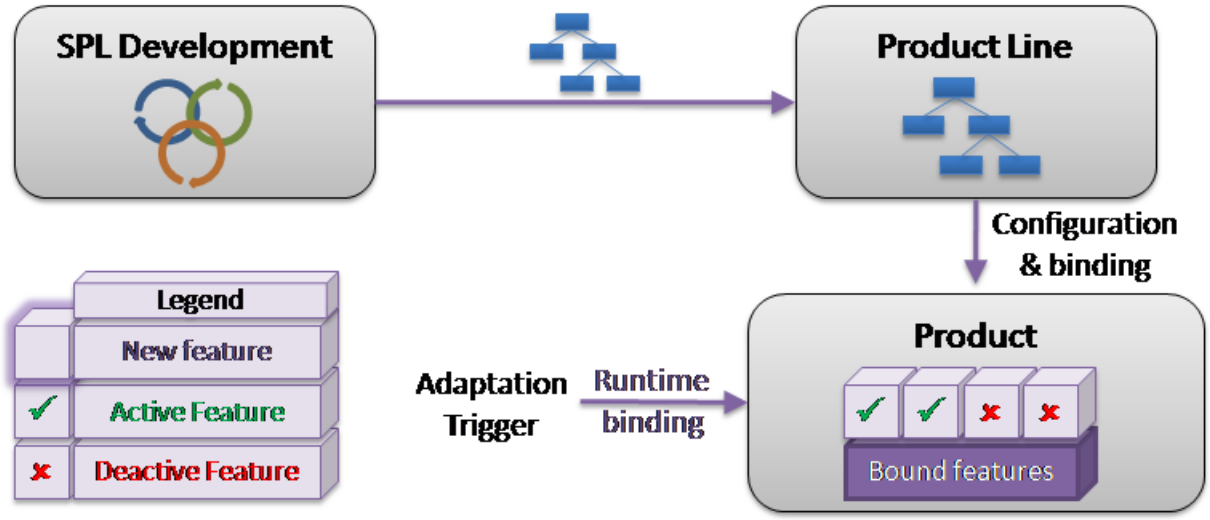

Figure 1. Product Life Cycle in Software Product Lines

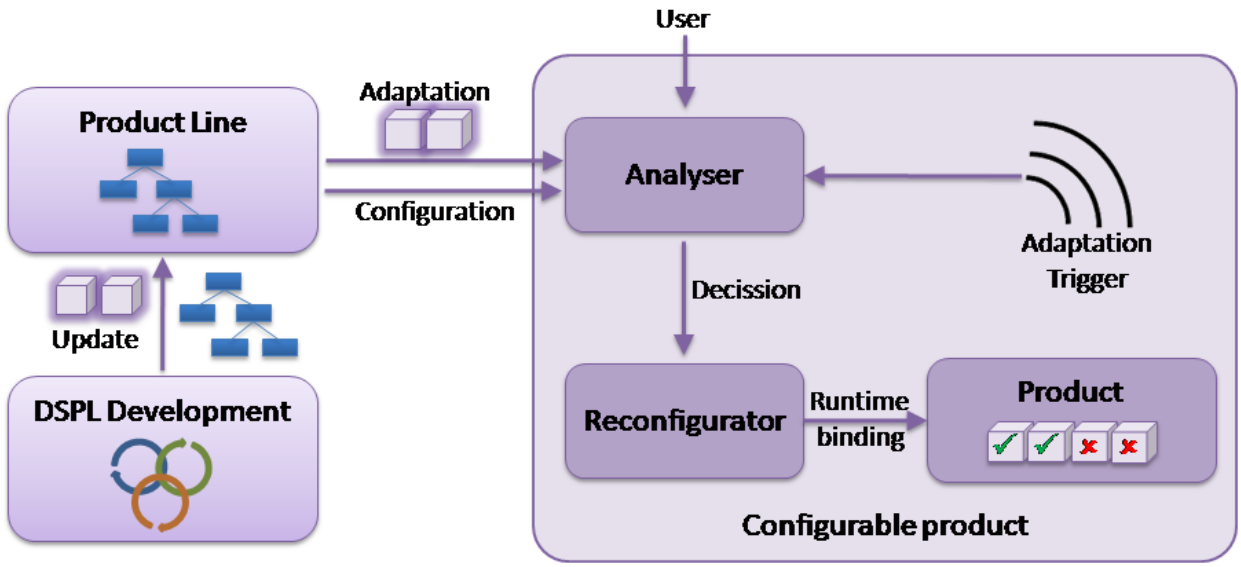

Figure 2. Product Life Cycle in Dynamic Software Product Lines

In the product perspective, we study the advantages and disadvantages obtained as result of incorporating adaptation in SPL products. The following criteria help us to evaluate the return of investment on DSPLs.

- **Adaptation capabilities.** This criteria addresses the adaptation level achieved by the SPL product.
- **Autonomic degree.** This criteria addresses how much products depend on SPL to perform adaptation.
- **Computational overload.** This criteria addresses how much computational overload is introduced by the DSPL approach.

Next, we apply these criteria to both connected and disconnected DSPLs and then we illustrate how they achieve product adaptation.

3.1 Connected DSPL

Connected DSPLs stay in touch with products in order to send them updates. These updates enable products to deal with context changes. Figure 3 shows the steps to send the updates from the DSPL to the CPs.

1. The CP senses a relevant change which starts the adaptation process. Both changes in the environment and in the CP itself can trigger the adaptation process.
2. The CP sends information about the change to the SPL. Optionally, the CP can preprocess the information locally and then it sends a more specific information to the SPL.
3. The SPL incorporates the acquired information to the product requisites and then it calculates a new CP variant.
   (a) If there is no variant that satisfies the product requisites, then the SPL notifies the CF and the adaptation process fails.
4. The SPL generates the update for the CP. The update can be the whole calculated variant or the difference between the old variant and the new one.
5. The SPL sends the update to the CP.
6. The CP updates itself using the update from the SPL.

According to the criteria presented before, we characterize a connected DSPL as follows:

- **Adaptation mechanism.** The CP gets **updates** from the SPL to perform the adaptation.
- **Contact between the SPL and the product.** It is necessary a bidirectional connection between the DSPL and the CP. If this connection becomes unavailable then the adaptation can not be performed.
- **Variability manager.** Only the SPL is in charge of manage variability.
- **Adaptation capabilities.** The more scope address the SPL, the more adaptable the CP is.
- **Autonomic degree.** The CP depends on the SPL to perform the adaptation. However, if the connection is unavailable, the CP behaves as a classic product.
- **Computational overload.** The product overload depends of (1) the communication with the SPL and (2) the online installation of updates.

3.2 Disconnected DSPL

Disconnected DSPLs produce CP which can reconfigure itself to deal with contextual changes. Compared with connected DSPLs, CP reconfigures itself without any DSPL contact. CP are augmented with variability knowledge and quiescent components in order to perform the reconfiguration as Figure 4 shows:

1. The CP senses a relevant change which starts the adaptation process. Both changes in the environment and in the CP itself can trigger the adaptation process.
2. The CP calculates a new configuration to deal with the sensed change.
   (a) If there is no configuration that satisfies the product requisites, then the adaptation process fails.
3. The CP reconfigures itself to apply the calculated configuration. The reconfiguration operation implies (1) start/stop components and (2) establish connections between them.

According to the criteria presented before, we characterize a disconnected DSPL as follows:

- **Adaptation mechanism.** The CP **reconfigures** itself to perform adaptation.
- **Contact between the SPL and the product.** There is necessary no connection between the SPL and the CP. The adaption depends on CP resources only.
- **Variability manager.** Only the CP is in charge of manage variability.
- **Adaptation capabilities.** The more variability knowledge the CP have, the more adaptable the CP is.
- **Autonomic degree.** The CP have no dependency of the SPL to perform the adaptation.
- **Computational overload.** The overload is depends of (1) the variability analysis and (2) the online reconfiguration.

3.3 DSPL State of the Art

Figure 5 shows the methods included in connected and disconnected categories. They have been ordered according to year in which they appear in the literature.

According to Figure 5, there are six DSPL that focus their efforts on developing configurable products. An overview of these DSPL is presented next:

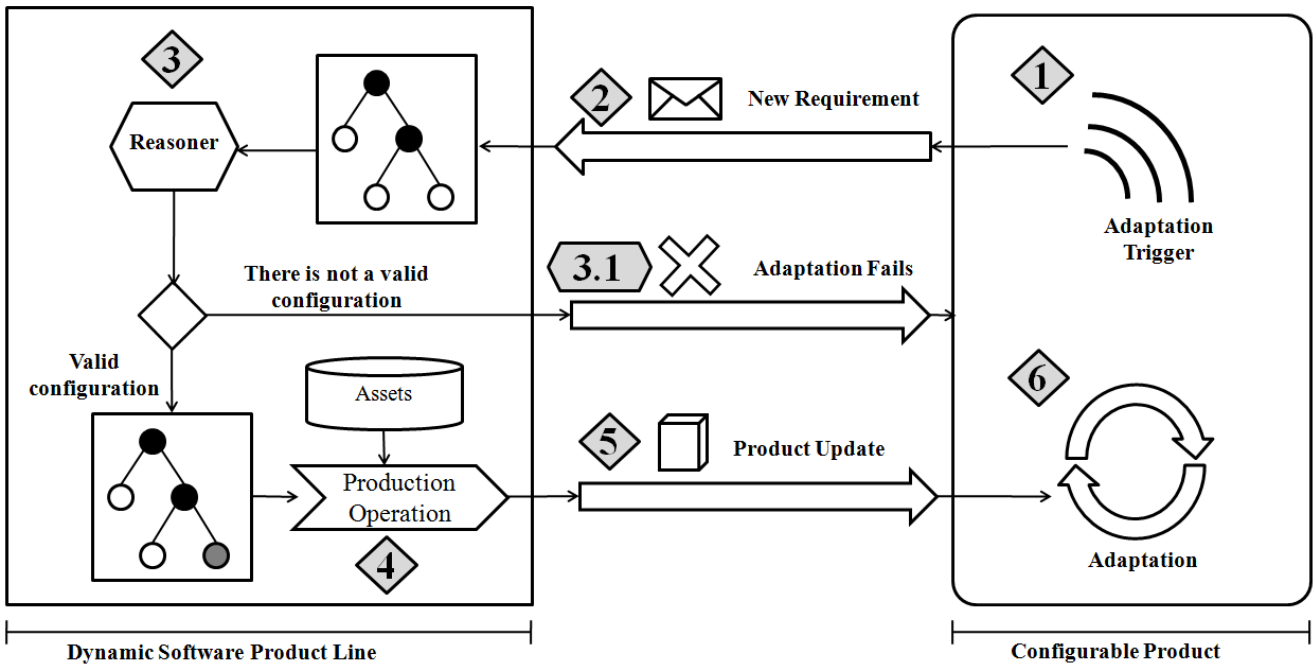

Figure 3. Connected DSPL Overview

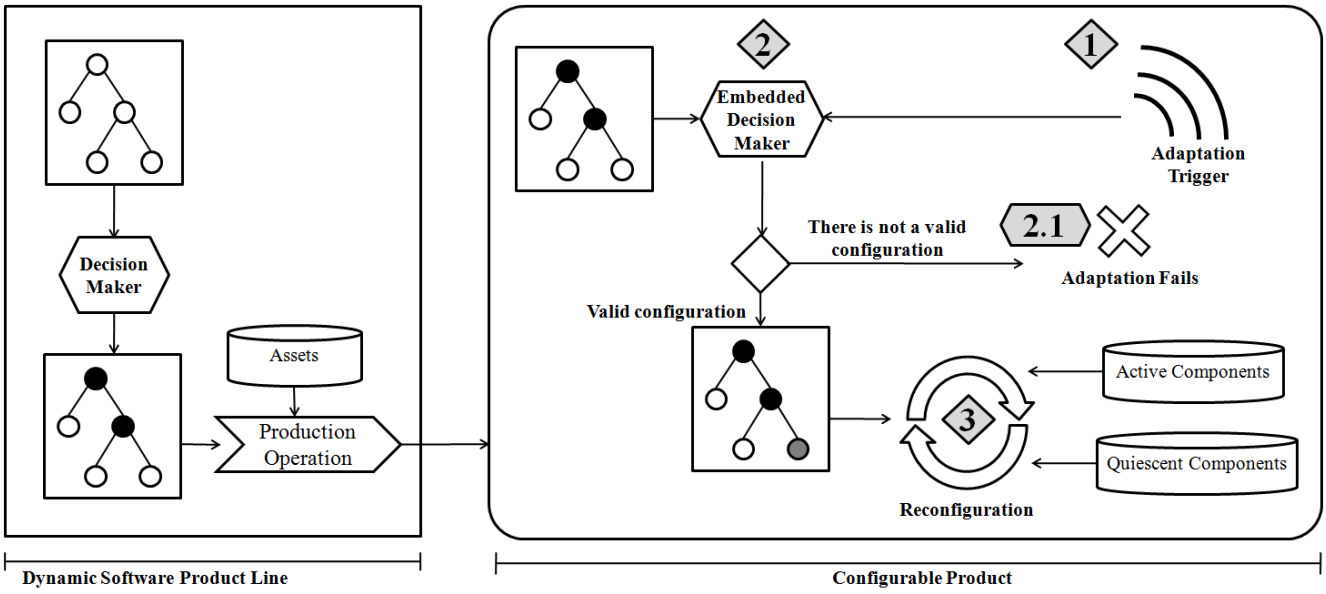

Figure 4. Disconnected DSPL Overview

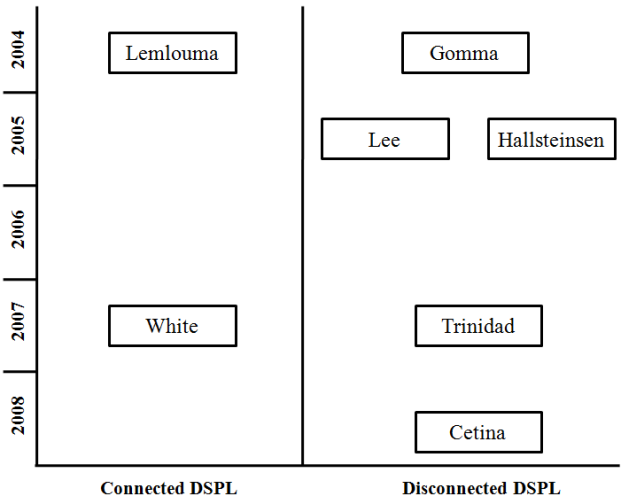

Figure 5. Classification of DSPL

- **Gomaa.** Reconfigurable Product Line UML Based Environment (RPLUSEE) [3] was proposed by Gomaa et al. Their main contribution is provisioning software dynamic reconfiguration patterns. Depending on the location of dynamic reconfiguration information, these patterns are classified into master-slave, centralized, client-server and decentralized. This method also provides reconfiguration Statechart and reconfiguration transaction models for the dynamic reconfiguration. This approach focuses on high-level specifications of dynamic reconfigurable units; however, it does not describe techniques and guideline for for reconfigurable component identification, design and implementation detail.
  Adaptation Trigger. Users specify runtime configuration changes so that executable system is dynamically changed from the old configuration to the new configuration.

- **Lemlouma.** Lemlouma et al. [7] present Negotiation and Adaption Core architecture for adapting and customizing content before delivering it to a mobile device. Their strategy takes into account device preferences and capabilities which are specified using a device independent model. These models are queried using the XQuery language and the adaptation is achieved by means of client repositories and SOAP services.
  Adaptation Trigger. There is a real time evaluation of the context dimensions. Lemlouma explicitly identifies the following context dimensions: user preferences, network speed and current confection protocol.

- **Lee.** Lee et al. proposed a systematic method to developing dynamically reconfigurable core assets and a reconfigurator that monitors and manages product configuration at run-time [6]. The method first analyzes a product line in terms of features and their binding time. Then, core assets are developed with the analysis results as key design driver. Finally, the developed reconfigurator address reconfiguration contexts, reconfiguration strategies and reconfiguration actions (what to do to reconfigure).
  Adaptation Trigger. The configuration plane is in charge of detecting contextual changes. The plane consists of two components: *Master Configurator* and *Local Configurator*. *Master Configurator* collects information from *Local Configurators* and/or external probes to detect contextual changes. Each *Local Configurator* monitors local messages within the product.

- **Hallsteinsen.** The MADAM approach [5] was developed by Hallsteinsen et al. This approach builds adaptive systems as component based systems families with the variability modeled explicitly as part of the family architecture. MADAM uses property annotations on components to describe their Quality of Service. For example a Video Streaming component may have properties such as start up time, jitter and frame drop. At run-time, the adaptation is performed using these properties and a utility function for selecting the component that best fits the current context.
  Adaptation Trigger. The *Context Manager* is responsible for managing and monitoring a set of contexts in the system environment relevant for the adaptation. Context includes execution platform context elements such as network and memory resources, the environment context elements such as light and noise, and user context elements location and stress level.

- **White.** The Scatter tool [12] was developed by White et atl. to address efficient online variant selection. Scatter captures the requirements of the product line architecture and the resources of a mobile device and then quickly constructs a custom variant for the device. This tool also ensures that variant selection is optimal with regard to a configurable cost function.
  Adaptation Trigger. A mobile device discovery service obtains the non-functional properties of a devices such as *JVMVersion* or *Position*. This service is implemented using SOAP-based web service and a CORBA remoting mechanism for remotely communicating device characterizations as they are discovered.

- **Trinidad.** Trinidad et al [9] present a process to automatically build a component model from a feature model based on the assumption that a feature can be modeled as a component. This process focuses on enabling a dynamic SPL to dynamically changing a

product by activating or deactivating its features at runtime.
Adaptation Trigger. One o more users set the requirements of the product.

- **Cetina.** Cetina et al. proposed a DSPL based on Model Driven Development and Variability Modeling principles [2]. Variability models are interpreted at run-time to reconfigure pervasive systems according to fluctuations in the environment. These model introduce precalculated reconfigurations to speed up adaptation in failure scenarios. The approach improves DSPLs to produce software that adapts itself in an autonomic way.
Adaptation Trigger. Reconfiguration is triggered by means of resources changes such as a new resource is installed or a resource becomes unavailable. These resources are sensors, actuators and external software services. Reconfiguration is also triggered when users explicitly enable/disable a product functionality.

Table 1 summarizes the specification techniques of each approach as well as the underlying infrastructure to support adaptation.

4 Our proposal: Mixed DSPL

On the one hand, connected DSPLs produce highly adaptable CP. However, adaptation implies that CPs must stay connected with their DSPL. On the other hand, disconnected DSPL produce autonomic CPs which have and adaptation range shorter, but they do not need to stay connected with a DSPL in order to adapt itself.

To bridge the gap between connected and disconnected DSPLs, we propose an hybrid approach called mixed DSPL. Mixed DSPLs produce scenario aware CP. In involution scenarios, CPs behave as in a D-DSPL while, in evolution scenarios CPs behave as in a C-DSPL. Figure 6 shows the steps performed by mixed CP in order to perform adaptation.

1. The CP senses a relevant change which starts the adaptation process. Both changes in the environment and in the CP itself can trigger the adaptation process.
2. The CP calculates a new configuration to deal with the senses change.
   (a) If there is no configuration that satisfies the product requisites, then the CP delegates the adaptation to the SPL. Therefor, the CP sends information about the change to the SPL. Optionally, the CP can preprocess the information locally and then it sends a more specific information to the SPL.
   (b) The SPL incorporates the acquired information to the product requisites and then it calculates a new CP variant.
      i. If there is no variant that satisfies the product requisites, then the SPL notifies the CF and the adaptation process fails.
   (c) The SPL generates the update for the CP. The update can be the whole calculated variant or the difference between the old variant and the new one.
   (d) The SPL sends the update to the CP.
   (e) The CP updates itself using the update from the SPL and the adaptation process ends..
3. The CP reconfigures itself to apply the calculated configuration. The reconfiguration operation implies (1) start/stop assets and (2) establish connection between them.

According to the criteria presented before, we characterize a mixed DSPL as follows:

- **Adaptation mechanism.** The adaptation mechanism depends of the dynamic scenario. In involution scenarios the CP applies the **reconfiguration** mechanism, and in evolution scenarios the CP applies the **updates** mechanism.
- **Contact between the SPL and the product.** The DSPL-CP connection is not strictly necessary to achieve some level of adaptation. Although, DSPL-CP connection is necessary to achieve fully adaptivity.
- **Variability manager.** Both the DSPL and the CP are in charge of manage variability.
- **Adaptation capabilities.** The more scope address the SPL, the more adaptable the CP is.
- **Autonomic degree.** Some level of adaptivity is guaranteed, even if the DSPL-CP connection is unavailable.
- **Computational overload.** In the worst case, the overload is depends of (1) the local variability analysis, (2) the communication with the SPL and (3) the online installation of updates.

The duality of mixed DSPLs introduces new challenges to DSPLs developers. Next sections describe the key infrastructure elements of mixed DSPLs.

	Variability Specification	*Adaptation Infrastructure*
Gomaa	UMLS State Charts	Reconfiguration Patterns
Lemlouma	Device Independent Model	SOAP Services
Lee	Feature Models with Binding Units	Dynamically Reconfigurable Assets
Hallsteinsen	UML QoS Profile Model Properties	Planning Process
White	Requirements and Resources Specification	Variant Delivery
Trinidad	Feature Model	Relationship Components
Cetina	Scope, Commonalities and Variability Model	OSGI Service

Table 1. DSPL Comparison

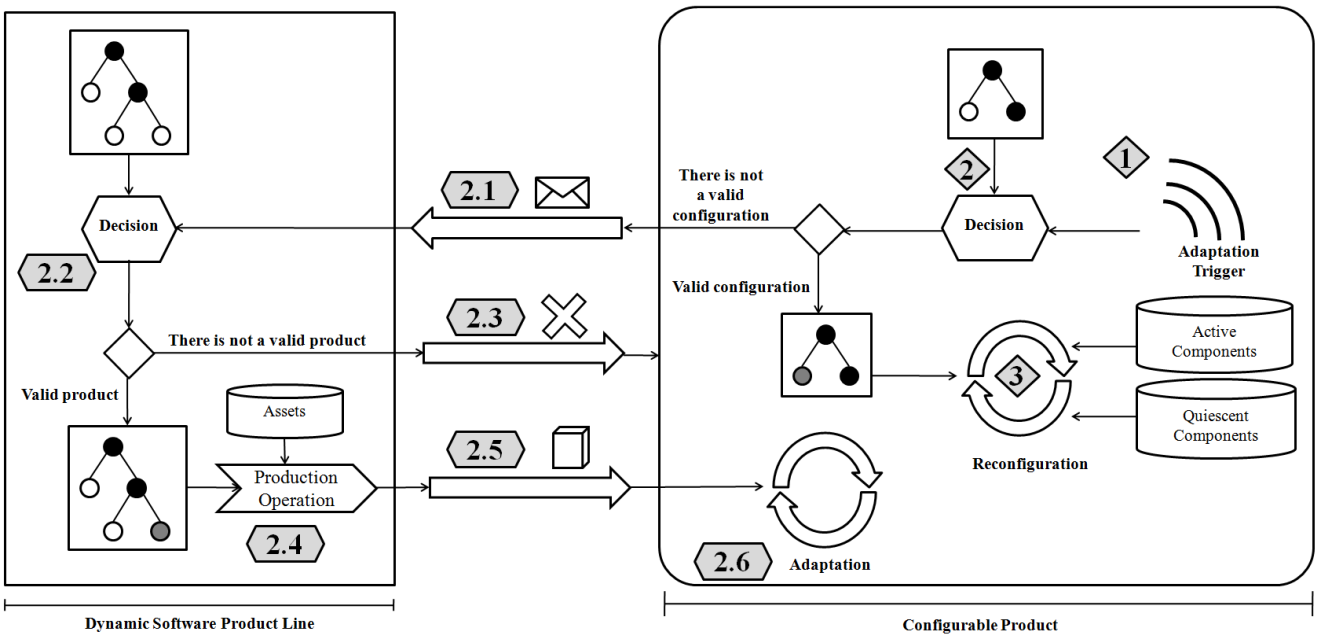

Figure 6. Mixed SPL Overview

4.1 Configurable Product Generation

The production of a CP differs a bit from common SPL configuration, where features are selected or removed to produce a final product. A CP might be considered as a partial or staged configuration of a variability model where three kinds of feature appear:

1. Discarded features: feature that are not deployed in a CP.
2. Active features: features that are deployed and activated in a CP.
3. Deactive features: features that are deployed in a CP but are not activated, however they are available for reconfiguration.

DSPL use variability models to describe the derivable CP. To keep the information about the dependencies and relationships among CP features, an specific variability model may be used. A CP variability model is generated from the DSPL variability model where discarded features are removed. It is important to check both, the selection and the resultant CP variability model for containing no errors, such as mandatory features that have accidentally being removed or incompatible selections of features.

From a CP point of view, two kinds of features are considered in variability models: active and deactive features. Reconfiguration is held on the reconfiguration component that is in charge of de/activating the features to adapt itself when adaptation triggers are received.

4.2 Decision Maker

The decision maker is in charge of reacting to the adaptation triggers, deciding which features must be activated or deactivated. The way to make decisions is not fixed and

many alternatives may be considered such as rules system or *ad-hoc* reasoners that use logic paradigms. The information to be taken into account to make decisions is:

- The features available in the CP and their state (activated or deactivated)
- Dependencies among features.
- Adaptation triggers that inform about an involution scenario or a needed feature.
- User requests of features activation or deactivation.

Then, a variability model may be used to represent information relative to features and their relationships, and the adaptation trigger or user requests inform about a list of features to be activated or deactivated. Among the decision maker responsibilities that we have detected we mention:

- Giving a fast response to involution scenarios by means of precalculated reconfigurations.
- Asking the DSPL for new reconfigurations when an evolution or involution scenario is performed.
- Giving a response to evolution and involution scenarios whenever DSPL is not available.
- Communicating the reconfigurator for the features to be activated or deactivated in reconfigurations.

On the DSPL side, its decision maker is conscious of the whole structure of the DSPL by means of its variability model. Moreover, the SPL knows the big picture of the DSPL, its variability model manages more knowledge than a CP variability model and commonly pervasive systems are computationally more limited than other systems. Because of these two factors, we determine the following responsibilities of the SPL Configurator:

- Calculating the reconfiguration in involution and evolution scenarios and send them to the CP configurators.
- Generating a CP variability model from the SPL variability model and the selected features.

FAMA Framework [11] is an SPL able to produce customized tools to reason about variability models. It uses different logic paradigms and algorithms to reason and extract information from variability models to help on decision making.

Although any solution may be given to implement a decision maker, we propose using FAMA Framework to generate both general DSPL and CP decision makers. We have to determine which operations are needed to extract information from a variability model and make decisions:

- Producing a CP variability model from the SPL variability model either when a CP is firstly produced or in an evolution scenario.
- Calculating involution scenarios by propagating decisions when a feature is de/activated.
- Providing explanations [10] when a reconfiguration is not possible.

As FAMA Framework is an SPL itself, we may create specific products that only supports above operations at both sides. This is important in CPs whose resources are limited, as some functionality of FAMA Framework is not going to ever be used. In this case, we are currently working on developing an specific product for limited CP that we call FAMA Lite.

4.3 Elaborating a Contingency Plan

Evolution scenarios are predictable and commonly come from an updating process and may be scheduled. A new CP is regenerated where existing features are maintained and new ones are optionally added. The DSPL decision maker is in charge of deciding about the new CP configuration so the CP decision maker has no responsibility on this process but receiving and storing the new active configuration.

However, involution scenarios might arise from unpredictable events such as a device breakdown or an unavailable service. In these cases, a the fastest response is needed to restore an stable state of the CP where functionality is at least maintained or reduced to a minimum.

The CP reconfigurator must be in charge of determining the features to be activated or deactivated to take the CP to a stable state. However, explaining a failure and restoring it [10] is an NP-problem that in many cases might not produce a fast response. Failure restoring algorithms may be solved by using heuristics that return a response that may or may not be the best response. It will allow to give a response in a limited time at the risk of reducing the functionality of the CP more than it is needed due to a non-minimal diagnosis.

As the number of involution scenarios is finite and can be foreseen, a *contingency plan* may be built by the DSPL decision maker so the CP decision maker knows how to act in these cases, giving an optimum response and restoring the CP fastly to the state where most of the services are correctly working.

When an involution scenario happens, the contingency plan must be regenerated. In order to generate it as fast as possible, the CP decision maker may ask the DSPL for producing part of the contingency plan or a complete one. Whenever DSPL decision maker is not available, the contigency plan may be calculated on CP decision maker idle times.

5 Conclusions and Future Work

In this paper we have presented the state-of-the-art in DSPLs. We have classified these approaches in Connected and Disconnected DSPLS according to the CP adaptation mechanisms. To bridge the gap between connected and disconnected DSPLs, we have proposed an hybrid approach called mixed DSPL. Finally, we have discussed the underlying infrastructure to develop Mixed DSPLs.

Our future work aims at improving current decision makers to support mixed DSPLs. In concrete, we are working on the FAMA Lite support to deal with (1) three-state feature models and (2) contingency plans.

Furthermore, new challenges appear in the DSPL context from the automated analysis point of view such as the three-state variability models handling and determining semantic differences between analysis and runtime variability models.

Finally, we intend to develop mixed DSPLs to validate our proposal in a real context, concentrating our effort in smart houses.

References

[1] Software product-family engineering, 5th international workshop, pfe 2003, siena, italy, november 4-6, 2003, revised papers. 3014, 2004.

[2] C. Cetina, J. Fons, and V. Pelechano. Applying Software Product Lines to Build Autonomic Pervasive Systems. *Software Product Line Conference, 2008. SPLC 2008. 12th International*, 8-12 Sept. 2008.

[3] H. Gomaa and M. Hussein. Dynamic software reconfiguration in software product families. *Software Product-Family Engineering*, pages 435 – 444, 2004.

[4] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, April 2008.

[5] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. Using product line techniques to build adaptive systems. *Software Product Line Conference, 2006 10th International*, pages 10 pp.–, 21-24 Aug. 2006.

[6] J. Lee and K. C. Kang. A feature-oriented approach to developing dynamically reconfigurable products in product line engineering. *splc*, 0:131–140, 2006.

[7] T. Lemlouma and N. Layaida. Context-aware adaptation for mobile devices. *Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on*, pages 106–111, 2004.

[8] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, July 2004.

[9] P. Trinidad, , A. Ruiz-Cortés, and J. P. na. Mapping feature models onto component models to build dynamic software product lines. *International Workshop on Dynamic Software Product Line*, 2007.

[10] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6):883–896, 2008.

[11] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. Fama framework. In *Software Product Line Conference, 2008. SPLC 2008. 12th International*.

[12] J. White, D. C. Schmidt, E. Wuchner, and A. Nechypurenko. Automating product-line variant selection for mobile devices. *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 129–140, 10-14 Sept. 2007.