A Contract-based Approach for Managing Dynamic Variability in Software Product Line Architectures

Sergio T. Carvalho¹,², Leonardo Murta¹, Orlando Loques¹

¹Instituto de Computação – Universidade Federal Fluminense, Niterói, Brazil
²Instituto de Informática – Universidade Federal de Goiás, Goiânia, Brazil

{scarvalho, leomurta, loques}@ic.uff.br

Abstract. Software product lines capture commonalities and variabilities of reusable artifacts (like software components) to facilitate developing new products by exploiting these variabilities. However, variability management is traditionally done just in development time and deployment time, not attending classes of applications that require a high degree of adaptability such as ubiquitous/pervasive and context-aware applications. This technical report presents an approach for runtime variability management, considering product lines aimed at context-aware applications. The approach focuses on describing architectural contracts to deal with dynamic variability, and on their association with a software product line architecture description. Contracts allow describing adaptation services and context rules at a higher level of abstraction. Services encapsulate variabilities and are capable of adapting/reconfiguring the architecture of a product in a dynamic way, while the context rules determine the context in which a particular service should be activated. Our approach was applied in scenarios involving dynamic variabilities in a software product line focused on home health care applications, where it was possible to observe its feasibility.

1. Introduction

According to [Northrop 2002], the use of Software Product Line (SPL) techniques focuses on the design, development, and evolution of a family of related software products. All products share a common set of features and distinguish from each other by a set of variabilities. The creation of a specific product consists in a process called derivation, where the variabilities that should be present in the product are selected. Once all variabilities are resolved, including those related to required components, the product can be deployed...

¹This technical report is an extended version of the paper entitled “Dynamic Variability Management in Product Lines: An Approach Based on Architectural Contracts” [Carvalho et al. 2010b], that was indicated to the Best Paper Award of the 4th Brazilian Symposium on Software Components, Architectures, and Reuse (27-29 September 2010, Salvador, Brazil)
and run. This approach reduces the development time and costs [van Gurp et al. 2001] and improves quality in software product creation [Northrop 2002].

For example, when creating a product from a SPL aimed at the patient monitoring domain, one variability to be solved can be related to whether or not a “panic button” device should be used in the derived products. This equipment, usually worn by the patient as a pendant, sends an alarm to a supervisory medical center when activated. In many situations, however, its use may be unnecessary or even inappropriate, according to the patient care plan. Thus, the derivation of products with or without this device occurs depending on the specific characteristics of these products.

SPLs have proven to be an efficient way to handle the different needs of users [Hallsteinsen et al. 2006], since it is possible to customize a software product by selecting the variabilities before its deployment. However, this technique in its traditional form may not be convenient for products whose architectures need to adapt their configurations at runtime, such as context-aware applications known from ubiquitous computing [Huebscher and McCann 2008]. These applications have dynamic variability (resolved at runtime), requiring a SPL architecture with dynamic adaptation properties combined with mechanisms for context monitoring and controlling adaptations [Hallsteinsen et al. 2008]. In other words, the derivation process that traditionally takes place as a whole in development time occurs partially, producing a flexible product (sub product line), which allows further adaptations in the light of demands just occurring at runtime.

Applications for patient monitoring at home, for example, depend on the quality and availability of resources (e.g., sensors, devices, etc.), since these resources can be added, removed, or can simply fail. There are also issues related to the patient needs and preferences, which may change according to their health problems: a patient with hypertension, for example, can evolve to a patient with cardiac insufficiency, requiring new medical prescriptions and appropriate physiological sensors.

In this example, the presence of sensors and devices may require dynamic variability in the software, since these elements are realized only during the application execution. A product, designed and customized for the individual characteristics of a patient at home, must have an architecture capable of being adapted as a consequence of the dynamic variability management, according to the context, the environment, and the states of resources [Loques and Sztajnberg 2010]. For this purpose, a specialized supporting infrastructure should be available to continually check the context information from both the patient (e.g., location, health status, physiological data, activities, etc.) and the residence (e.g., map of the residence, distribution of sensors and devices in environments, environmental data, etc.).

This technical report presents a novel approach for dynamic variability management based on architectural contracts, which specify services that perform dynamic adaptation/reconfiguration of the architecture according to previously specified context rules. Contracts are associated with the SPL architecture description and encapsulate dynamic variabilities. In addition, they contain a negotiation machine that specifies the desired levels of quality and how these should be imposed on the architecture. A supporting framework performs context monitoring, services execution, and architecture adaptation.
control. Our approach differs from the existing dynamic adaptation approaches by promoting variabilities and their adaptations to a first-class element, encapsulated by contracts. This leverages the representational power of dynamic product lines and potentially benefits their evolution.

To demonstrate the feasibility of our approach, we present three scenarios based on dynamic variabilities observed in a SPL that is being developed as part of a project focused on the application of ubiquitous/pervasive computing in home health care systems. In the first scenario, our approach selects, during runtime, the most appropriate communication technology to connect the patient residence to a supervisory medical center, considering context information. Next, in the second scenario, our approach selects the closest and preferred device in the residence (e.g., TV, mobile phone, etc.) to notify the patient to carry out his/her medical prescriptions, also considering context information. Finally, in the third scenario, our approach performs dynamic adaptation of the product architecture according to the evolution of the patient’s health problems.

The technical report is organized as follows. Section 2 describes product lines and software architecture concepts that are important for the contextualization of this work. Section 3 presents a SPL for ubiquitous/pervasive home health care systems, focusing on dynamic variability and adaptation requirements found in this class of applications. Section 4 describes the proposed approach, and Section 5 presents how it deals with the previously discussed scenarios, considering the SPL introduced in Section 3. Section 6 mentions some related work, and finally, Section 7 presents some concluding remarks.

2. Product Line and Software Architecture

Many applications share similar features and concepts, establishing a family of related software products. This aspect, together with the need to reduce cost and effort and to increase quality in system development, led to the creation of SPL. The design of a SPL aims at building related products that satisfy the specific needs or mission of a market segment [Northrop 2002]. Similarities among products allow development to be undertaken considering a set of common reusable assets. Due to that, a SPL must capture similarities and differences among the products [van Gurp et al. 2001].

Similarities and differences can be represented through a feature model [Kang et al. 1990], where the similarities among the products of a SPL are known as commonalities (mandatory features), and the differences are known as variabilities, which are divided into three types: optional features, alternative features, and optional/alternative features. An optional feature may or may not be present in a product; an alternative feature is present, but should be selected from a set of alternatives; and an optional/alternative feature refers to an alternative that may or may not be present in a product. Feature model is a form of representing a domain, interrelating commonalities and variabilities. Furthermore, it is described at a high level of abstraction, focusing on variability management in order to derive (i.e., create) a specific (customized) product from the SPL.

Figure 1 shows a partial SPL feature model aimed at remote monitoring of patients at home (see Section 3 for a more complete model). In this model there are mandatory features (black circles) such as Decision Module capable of identifying the patient’s health situation, and Alarm that should report alarms to a supervisory center. There are also
optional features (white circles) such as physiological sensors to be used by the patient at home (*Blood Pressure*, for example). During the derivation of a specific product, the first two features should be present in the product, while the optionals (varibilities) may or may not be present in the product. If it is present, a blood pressure sensor can be selected, in case of a product customized for a patient with hypertension, or a heart rate sensor can be selected, if the product is target at a patient with cardiac insufficiency, or even both can be selected.

![Figure 1. Example of a SPL feature model. White circles identify variabilities and black circles identify commonalities. Notation used: [Kang et al. 1990].](image)

Besides the feature model, a SPL often includes a set of reusable components and a software architecture that describes how these components interrelate [Northrop 2002]. A product is created from the selection of variabilities that drives the configuration of components, according to the SPL architecture description, and applicable components are retrieved from the set of reusable components.

The SPL architecture provides abstractions to represent the structure and behavior of products. In general, a software architecture can be defined as a set of components, a set of connectors (interconnections between these components), and the organization of these elements (configuration) for a given software system [Medvidovic and Taylor 2000]; also, the specification of the system architectural elements can be made through an Architecture Description Language (ADL). Unlike the architecture designed for individual systems, the SPL architecture should represent the architectural elements corresponding to commonalities and variabilities (i.e., optional and alternative elements). Thus, a specific product architecture can be derived from the SPL architecture, incorporating common elements and selecting elements related to variabilities that should be present [Garg et al. 2003]. Figure 2 shows an architecture for the SPL feature model of the Figure 1. Specific product architectures can be derived with the *Blood Pressure Sensor* or with the *Heart Rate Sensor*, or also with both sensors.

Traditionally, variability selection in SPL is done only at the development time and deployment time of a product [Hallsteinsen et al. 2008]. This means that once variabilities are selected and the product is deployed, they cannot be changed or adapted during its operation. However, there are classes of applications that demand a higher degree of adaptability, such as ubiquitous/pervasive and context-aware applications. In such applications the variations in the environment, availability or quality of resources, or changes regarding the user needs, require responses from the application in terms of architectural reconfigurations at runtime.
3. A Software Product Line with Dynamic Variability

Ubiquitous/pervasive home health care applications, where patients are remotely monitored, requires a high degree of adaptability [Eslami and van Sinderen 2009, Loques and Sztaijnberg 2010]. Figure 3 shows the overview of a system that we are developing in the context of this application domain [Carvalho et al. 2011, Copetti 2010, Carvalho et al. 2010a, Copetti et al. 2009].

In this system, patient data (physiological and behavioral) and residential environment data are collected by sensors and processed by a Home Health Station (HHS). The collected data are represented by fuzzy variables and analyzed using artificial in-
telligence techniques. The eventual identification of an abnormal condition of the patient may activate a local device (a TV, for example), increase the frequency of monitoring, or, depending on severity, send an alarm to a Supervisory Medical Facility (SMF). Context-aware techniques are employed to provide access to context information, and perform the activity of discovering and monitoring resources (e.g., sensors, devices, etc.) [Sztajnberg et al. 2009]. A care plan guides the patient regarding measurements that must be done (e.g., blood pressure), medications that should be taken, and other personalized recommendations according to the treatment. Further details about this system are available in [Carvalho et al. 2011, Copetti 2010, Carvalho et al. 2010a, Copetti et al. 2009].

In general, current home health care systems have been proposed in a specialized manner without taking into account differences among patients, particularly regarding their preferences and needs, including those related to the disease under treatment [Eslami and van Sinderen 2009]. However, to be more effective, each system should be customized in terms of both patient and residence (map of the residence, distribution and type of the sensors and devices, etc.).

In this context, the product line approach allows system customization and, at the same time, development cost and effort reduction together with quality enhancements. Figure 4 shows a partial SPL feature model for the home health care domain. This model was built from the analysis of ubiquitous/pervasive home health care related work (e.g., [Orwat et al. 2008, EIHelw et al. 2009, Leijdekkers et al. 2007]), the knowledge gained from health experts, and the experience in development of a prototype aimed at the continuous identification of the patient’s health situation associated with a care plan [Carvalho et al. 2010a].

From this feature model, we can see that for a given patient at home, his/her monitoring requirements and software features necessary to their support can vary widely. This variation can occur during system deployment and system operation.

A custom system deployment in the residence must, in principle, specify a set of medical specialized rules for particular disease and an appropriate set of sen-
sors to gather relevant physiological and context data. In addition, a personalized care plan should be defined according to medical recommendations aimed at the patient [Loques and Sztajnberg 2010].

During system operation, in turn, changes in the patient’s health problems may require modifications in pre-configured resources (sensors, medical devices, etc.) or in the care plan, requiring system reconfiguration (dynamic adaptation). For example, the doctor, noticing changes in the monitored patient’s health, could indicate in the care plan that the patient uses a new device to measure his/her heart rate, and collect electrocardiogram data periodically. The system, originally customized for a patient with hypertension, should have its architecture dynamically reconfigured to meet this new demand (see scenario described in Subsection 5.3).

It is important to notice that, when considering the traditional use of SPL and in the light of the need for monitoring of a particular patient, a product is derived from the SPL and deployed into the patient’s home. The architecture of this product is fully attached to patient needs and his/her home characteristics at that point in time. If the patient’s health problems change in the future (e.g., from hypertension to cardiac insufficiency) or even if new devices are added to his/her residence, a new version of the product should be derived and deployed. The delay involved in this static assessment and the perception of the need for a new deployment could result in serious consequences for this class of system.

The care plan is a key element during system operation, since it can be seen as a sequence of tasks similar to a workflow and can be used for the interaction between the system and the patient. For example, the system can remember the patient, through the TV or mobile phone, that he/she needs to take a medication at a certain time. The choice of the interaction device can be done during the system deployment, but the residence may have different devices (TVs, HHS Display, etc.) as well as the patient may also have devices (mobile phone, tablets, etc.). In this situation, the choice of which device should show messages from the care plan may be delayed to the system execution and performed according to context rules, for example, choosing the device closest to the patient and that is his/her favorite (see scenario described in Subsection 5.2). The selection of the variability at runtime implies the dynamic adaptation of the system architecture in order to configure the chosen device.

System adaptation requirements also include the ability to deal with the availability of resources. For example, in the case of a fault in the communication channel between the HHS and the SMF, another available alternative communication can be deployed at runtime through dynamic adaptation of the product architecture (see scenario described in Subsection 5.1). This can be done transparently and without user intervention.

The SPL for home health care systems (Figure 4) has therefore variabilities that are resolved at runtime. These dynamic variabilities require mechanisms to provide adaptation/reconfiguration of the product architecture during its operation. The next section presents our concepts and mechanism to specify and implement our adaptation approach based on architectural contracts for dynamic variability management.
4. Dynamic Variability Management with Architectural Contracts

Our approach for runtime variability management relies on the description of architectural contracts to deal with dynamic variability, and on their association with a SPL architecture description. This way, a derived product architecture has associated contracts to handle with dynamic variabilities (Figure 5). In the software reuse jargon, contracts can be seen as configuration knowledge, which determines the required conditions for an adaptation and the effects of such adaptation.

![Figure 5. Contracts descriptions are associated with a SPL architecture description.](image)

Commonalities and variabilities selected at deployment time of a product are directly represented as architectural elements (components and connectors) and its dynamic variabilities, in turn, are described separately in contracts. During the product operation, services defined in the associated contracts adapt/reconfigure the architecture following context rules.

For example, a SPL can derive a product whose architecture has two components interconnected by a connector responsible for providing a communication technology. A contract can be created and associated with the SPL architecture, to encapsulate the decision of choosing the communication technology. This contract can be instantiated, during the product operation, in order to select and configure the most appropriate connector to that architecture, according to the available communication technologies, i.e., according to the context.

Contracts define adaptation services and context rules. Adaptation services describe dynamic variabilities while the rules determine a context in which a particular service should be performed. In our approach, contracts are written in CBabel [Cardoso et al. 2006, Loques et al. 2004], an ADL with features to describe software architectures and contracts, and mapped to a configuration support framework [Cardoso et al. 2006, Loques et al. 2004] that monitors the environment and effectively performs adaptations/reconfigurations. CBabel incorporates concepts from [Frølund and Koistinen 1998], which were reformulated for the context of software architecture descriptions.

Figure 6 shows the contract general structure. Each Adaptation Service is de-
scribed with architectural primitives capable of dealing with the dynamic variabilities (lines 03-15). The execution of these services is controlled by a Negotiation clause (lines 17-20), and occurs only if the context rules, defined in Profiles (line 06 and line 13), are met. These context rules are described using required context information, which are structured in one or more Categories.

When the contract is instantiated, the negotiation begins evaluating the service1 (line 18). The architectural primitives described in service1 are executed depending on the context rules in profile1 (line 06). If a rule is not met, a transition occurs to service2 (line 18). It is then evaluated in a similar manner. It is important to note that the service1 is defined as preferred service, since it is described in the left side of the first line of the negotiation clause. This means that service1 can be established by the configuration support framework at any time during the product operation, even though service2 is running. For the cases where no service meets the context rules, the no_service state is reached (line 19) and the configuration support framework can then initiate a new negotiation.

A typical contract has, therefore, four elements, namely, categories, profiles, adaptation services, and a negotiation clause (see descriptions below in the text). The definition of a contract in BNF notation is shown in Figure 7.

- **Categories:** describe at a high abstraction level properties of resources associated to the context where the product executes (e.g., location of devices such as TV and mobile phone, sensor types, communication types, etc.). For example, Device may be a category, and location a property of this category.
Categories are associated to the framework’s entities that allocate and monitor their properties. It worth noting that properties of resources, described in categories, can be used to specify semantical relationships among different kinds of resources [Ranganathan et al. 2005], establishing ontologies which can be used by resource discovery service.

- **Profiles**: quantify and establish values to the properties of a given category. The quantification constrains each property according to its description, acting as an instance of acceptable values for a given category. For example, the statement `Device.location = %currentPatientLocation` means that, at runtime, the current patient location at home (e.g., in the bedroom) is the only acceptable value for the `location` property of the `Device` category, thus defining a context rule.

- **Adaptation Services**: define constraints to be applied to the product in terms of its architecture, through one or more CBabel primitives (e.g., `instantiate` and `link`). The application of these constraints leads to dynamic adaptation/reconfiguration of the product architecture, accomplished by associating one
or more profiles. For example, a component corresponding to a dynamic variability, can be instantiated and linked to the product architecture according to the example profile shown above, i.e., only if the location of a device in the residence is the same as the patient (e.g., TV and patient are in the bedroom). Each adaptation service defines a possible operational state for the product.

- **Negotiation**: describes a policy defined by a state machine that establishes a particular order to deploy the adaptation services. According to a policy described in the negotiation clause, when a higher preference adaptation service cannot be maintained with the same constraints defined in the profile, the framework tries to deploy a service with lower preference. The return to a service of higher preference can also be described, allowing a service with better quality to be deployed if the associated profiles become valid.

Additional aspects about the contract semantics are available in [Braga et al. 2009]. They present an operational semantics for contracts in Cbabel and its implementation in a prototype tool, which allows execution and analysis of contracts.

Figure 8 shows, in the form of a UML class diagram, the relationship among contract elements, and the relationship between a contract and a SPL architecture. A contract is composed of adaptation services with their profiles, and a negotiation. A adaptation service is formed by profiles that are formed by categories. A negotiation, in turn, has adaptation services in its structure. It is worth noting that contracts are just associated with the SPL architecture, and can be created therefore independently of it.

![Figure 8. UML class diagram showing the relationship among contract elements, and between a contract and a SPL architecture.](image)

The semantics of a contract, written in Cbabel, is enforced at runtime by the configuration support framework. It provides, among other resources, context monitoring and adaptation control, central tasks in a Dynamic Software Product Line (DSPL) [Hallsteinsen et al. 2008]. The main functions of the framework are: (i) parse the contract specifications and store them as meta-level information associated to the product;
(ii) provide reflective and adaptive mechanisms, which allows adapting and managing the architectural configuration, in order to cope with the contract demands; and (iii) provide a set of mechanisms to interpret, impose, monitor, and manage these contracts associated with the product. Furthermore, the framework integrates context, monitoring, and resource discovery services.

The information flow among the main elements of the configuration support framework is shown in Figure 9.

Figure 9. Information flow among the main elements of the configuration support framework.

**Context Service** provides context information and continually monitors the context properties described in the profiles. It obtains the individual context information from each of the properties and, after collecting the information, it sends the consolidated result to the **Contract Manager**. For example, **Context Service** sends, during the product operation, the location of a device in the residence (e.g., TV is in the bedroom).

**Contract Manager**, in turn, is responsible for selecting services and for managing policies described in contracts. It verifies the constraints (context rules) required by the profiles. If the constraints are met, the corresponding adaptation service is selected to run. Otherwise, another adaptation service should be selected according to the negotiation clause. The same occurs if, during the operation of a given adaptation service, the context changes and the profile becomes invalid. In this case, the service cannot be maintained, and it is therefore terminated. For example, an adaptation service that deploys and links a TV component is selected to run only if the TV and patient are in the same room of the residence (e.g., in the bedroom).

Finally, the **Configurator** element maps the contract descriptions (in CBabel) corresponding to the selected adaptation service, into actions that carry out adaptations/reconfigurations of the product architecture.

More details about these and the other elements, and about the operation of the framework, are available in [Cardoso et al. 2006, Loques et al. 2004].
5. Scenarios

During our research we have developed some scenarios related to the SPL for home health care systems, with the aim of demonstrating the feasibility of this approach. Here we present three of these scenarios to illustrate and clarify our approach. The first scenario deals with communication between the HHS (Home Health Station) and SMF (Supervisory Medical Facility). The second scenario, in turn, selects a device (TV, mobile phone, etc.) closest to the patient and of his/her preference, in order to send messages to remind him/her of carrying out the medical prescriptions. Finally, the third scenario copes with modifications in the needs of the patient arising from changes in his/her health problems.

5.1. Scenario 1: Communication

Emergency data transmission from HHS to SMF may suffer interruptions, usually associated with the quality of service delivered by access providers, since the communication is done mainly via the Internet “commodity”. In this sense, we can provide fault tolerance features using an alternative channel through the cellular network or even via landline. Figure 10 shows a SPL architecture and a contract illustrating the communication feature.

<table>
<thead>
<tr>
<th>Internet</th>
<th>HHS</th>
<th>SMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cellular</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dotted connectors (Figure 10a) correspond to the alternatives communication feature, which are available as communication technologies that can be selected for the interconnection between HHS and SMF components. Figure 10b presents a SPL architecture.
description, in which HHS and SMF components (lines 02-03) are defined. The connector between the components must be dynamically established by the contract available in Figure 10c.

The contract (Figure 10c) defines three different services, where each service describes a communication alternative (lines 06-09, 10-13, and 14-17). The support required for each service is encapsulated in a connector (link statement), following context rules defined in the profile, which is shortly determined by the Communication.technology property. There is a communication connector for each channel option and a specific sensor that detects whether the corresponding channel is available or not.

Transitions between services depend on the availability of these services. The negotiation clause (lines 18-22) states the order in which the services are evaluated, and the line 19 (first line) determines that the preferred service is sInternet. If this service is not available, the sLandline service runs. If the sInternet becomes available again it is re-established by the configuration support framework, specifically through the element Contract Manager, even if the sLandline service is running. Line 20 imposes the same mechanism, but employing the sLandline and sCellular services. Finally, if the sCellular service is not available, neither service can be established (no_service). In this case, the configuration support framework starts a new negotiation, and this no_service state is communicated to the operators at the SMF in order to carry out procedures to ensure patient care. For this purpose, as a backup, the SMF includes a mechanism to monitor the connectivity of the HHS.

Figure 11 shows two possible configurations for the product architecture, based on the SPL architecture, and on the associated contract. To establish the communication link, the configuration support framework sets its connector and try to keep the link on the preferred channel defined by the contract (Figure 11a). However, if the Internet fails during the product operation, it is necessary to replace the connector in order to maintain the communication link. According to the contract, the adaptation service sLandline should add another connector corresponding to Landline, resulting in the product architecture shown in Figure 11b. Thereafter, the configuration support framework tries to re-establish Internet connector again, according to the negotiation clause of the contract.

![Figure 11. Product architecture operating with: a) Internet connector; and b) Landline connector.](image)

This scenario presents the usage of contracts in the managing of variabilities related to non-functional aspects, in this case, communication resources availability. Other
non-functional aspects can be considered in this SPL architecture, e.g., reliability, associating an encrypted communication protocol to a connector on the product architecture.

5.2. Scenario 2: Context-Aware Notification

The second scenario deals with optional features related to devices such as TV, mobile phone, HHS Display, etc., which can be used to notify the patient to carry out the medical prescriptions such as taking a drug, performing a measurement with a physiological sensor (e.g., blood pressure, heart rate, etc.), and other personalized recommendations according to the treatment. These prescriptions are maintained by a care plan that must interact with a notifier, which sends messages to devices available at the patient’s home. The devices are SPL features and, in this scenario, are represented as dynamic variabilities. Thus, the selection of devices to show messages to the patient must be made at runtime and according to context rules. Due to that, a SPL architecture can be described with the Care Plan and Notifier components, once they are SPL commonalities, and with a connector linking them (Figure 12). Lines 02-03 describe these components and the line 04 describes the interconnection (Figure 12b). HHS Display, considered mandatory here, is described and interconnected in lines 06-07.

![Figure 12. A SPL architecture for the patient context-aware notification.](image)

The SPL architecture also provides components, for TV and Mobile Phone, and interconnection between these devices and the Notifier. Part of this architecture, represented by dotted components and connectors, should be associated with contracts used to allow the devices selection at runtime.

In this scenario, TV instantiation (selection) depends on a specific context: it should be near to the patient. The same context rule applies to the Mobile Phone, while
the HHS Display does not depend on any context rule, and thus it is always instantiated. Figure 13 shows an example of product architecture, where HHS Display is used to show notification messages to the patient.

![Figure 13. Product architecture operating with the HHS Display.](image)

The product architecture configuration will be done by connecting devices to the Notifier, according to the patient preferences described in a profile. The patient will receive notification messages in his/her preferred device from among those located closest to him/her (e.g., in the same room of the residence). Two categories were defined in Figure 14a, Patient and Device, with context properties to be evaluated for services of a contract associated with the SPL architecture. Values to be queried by the context service are defined in two profiles, nearestDeviceProfile and preferredDeviceProfile, shown in Figure 14b. The last one (lines 17-20) is a special type of profile, capable of selecting only one of the instantiated components according to selection criteria. In this scenario, a precedence rule defines the TV as the preferred device of the patient (line 19).

Figure 15 shows a contract capable of selecting devices (TV and/or mobile phone) that meet the nearestDeviceProfile profile, i.e., devices that are in the same room of the patient.

The contract describes three services to resolve these variabilities corresponding to the devices (lines 04-17): the first service (lines 04-09) verifies if both devices (TV and mobile phone) meet the nearestDeviceProfile profile; the second service (lines 10-13) verify if only TV meets the same profile; and the third service (lines 14-17) verifies if only mobile phone meets also the same profile. In all cases, if a device does not meet the profile, is turned off, or not operational, the service will not start. The negotiation clause defines an order to start the services. If no_service state is reached (line 21), it means that there are no devices in the same room of the patient.

In addition, this contract will receive in its instantiation two numerical context variables (described in the two first lines), used to identify, respectively, the patient (patientId) and his/her current location at home (patientLocation). A location service keeps track of these variables and can be integrated with the configuration support framework [Sztajnberg et al. 2009].

The statement instantiate instantiates (selects) the device only if the patient
located by the location service (patientId) is the patient registered in the system, i.e., if Patient.id = %patientId (line 13 of the nearestDeviceProfile profile), and if he/she is in the same room (patientLocation) as the device is being instantiated, i.e., if Device.location = %currentPatientLocation (lines 14-15 of the same profile). If these context rules are not met, the device instantiation (selection) will not be performed.

The tvAndMobilePhoneService service of Figure 15 can select more than one device that is in the same room as patient is. For example, if the patient is in the bedroom and both the TV and the mobile phone are also in the bedroom, the service will select both devices. In addition to these contextual information, it is therefore relevant to consider information related to patient preferences, essential in home health care applications. Figure 16 presents a contract to connect the patient’s preferred device and the notifier.

Statement select (lines 04-05) makes the configuration support framework locate Device class instances corresponding to the devices closest to the patient. After, it selects among these devices (Device@room) one preferred by the patient, according to the context rules defined in the selection profile preferredDeviceProfile (lines 18-19 of the Figure 14b). The selected device (@device) is then connected to the notifier, causing the adaptation of the product architecture.

If the patient is therefore in the bedroom, the notification messages will be displayed on the TV because it is his/her preferred device. However, if it is turned off or not operational, messages will be displayed on the mobile phone (lines 14-17 of contract of

```plaintext
01 category Patient {
02    id: numeric in;
03    name: string in;
04    location: numeric in;
05
06 }

07 category Device {
08    location: numeric in;
09    type: enum (TV, MobilePhone, HHSDisplay) out;
10  }

(a)

12 profile {
13    Patient.id = %patientId;
14    Device.location = %currentPatientLocation;
15  }

16  nearestDeviceProfile;

17 selection {
18    Device.type = (TV > MobilePhone > HHSDisplay);
19  }

20  preferredDeviceProfile;

(b)

Figure 14. a) Patient and Device categories; b) nearestDeviceProfile and preferredDeviceProfile profiles.
```
Figure 15. Contract to select device closest to the patient.

```
contract nearestDeviceContract (patientId,
    patientLocation) {
  context @room
  service {
    instantiate Device as tv
    with nearestDeviceProfile;
    instantiate Device as mobilePhone
    with nearestDeviceProfile;
  } tvAndmobilePhoneService
  service {
    instantiate Device as tv
    with nearestDeviceProfile;
  } tvService
  service {
    instantiate Device as mobilePhone
    with nearestDeviceProfile;
  } mobilePhoneService
  negotiation {
    not tvAndmobilePhoneService -> tvService;
    not tvService -> mobilePhoneService;
    not mobilePhoneService -> no_service;
  }
}
```

Figure 16. Contract to select the preferred device among those closest to the patient.

```
contract preferredDeviceContract {
  service {
    link notifier to @device
    @device=select (preferredDeviceProfile,
      Device@room);
  } preferredDeviceRoomService;
  negotiation {
    not preferredDeviceRoomService -> no_service;
  }
}
```

the Figure 15). Figure 17 shows the resulting product architecture when TV is selected. The HHS Display will be chosen only if no device is near to the patient; as the example, if he/she is not in the bedroom, but in another room of the residence.

According to the SPL architecture, categories, profiles, and contracts descriptions, the patient will receive notifications via the device that is closest to him/her, respecting his/her usage preferences. More details about the resource discovery mechanism and dynamic binding are available in [Sztajnberg et al. 2009, Cardoso et al. 2006].

5.3. Scenario 3: Changes in the Patient’s Health Problems

This third scenario deals with the dynamic adaptation caused by changes in the patient’s health problems. Figure 18 shows a SPL architecture with elements corresponding to the commonalities Decision Module and Alarm, and corresponding to the variabilities Hypertension Rules, Accelerometer, Blood Pressure Sensor, Cardiac Insufficiency Rules, Heart Rate Sensor and Weight Scale.

In this architecture, the Decision Module identifies the patient’s health status based on Hypertension Rules and/or Cardiac Insufficiency Rules, defined in their respective components. These components, in turn, are interconnected to the appropriate sensors according to the patient’s health problem: patient with hypertension uses Accelerometer and Blood Pressure Sensor, and patient with cardiac insufficiency uses Heart Rate Sen-
Figure 17. Product architecture operating with the TV.

A product architecture should be structured with the common elements described in the SPL architecture (Figure 19a). The elements corresponding to variabilities depend on the evolution of the patient’s health problems, and should be resolved at runtime according to contracts associated with the SPL architecture. Figure 19b shows a contract built to deal with dynamic variabilities related to the needs of patients with hypertension.

The service hypertensionService (lines 07-17) adapts the product architecture, adding sensors (accelerometer and bloodPressure) and rules related to hypertension (hypertensionRules). Nevertheless, this adaptation occurs only when the configuration support framework verifies in the care plan if the patient is hypertensive. This is done through hypertensionProfile profile (lines 9, 11 and 13).

Figure 20 shows the CarePlan category and the hypertensionProfile profile. The CarePlan category maintains properties related to the patient information, which are monitored by the configuration support framework. The hypertensionProfile profile sets a context rule associated to the healthProblem property (line 09): health problem defined in the care plan must be equal to “Hypertension”. Thus, the configuration support framework continuously monitors health problems described in the patient care plan, making the adaptation service sensitive to any changes that may be made by the doctor.

Figure 21 shows the product architecture adapted in order to provide resources needed to take care of a patient with hypertension. It uses sensors to the collection of his/her blood pressure and activities, and appropriate rules to identify emergency situations and generate alerts that can be sent to the supervisory center.

Nevertheless, the data analysis together with the warnings issued by the system may reveal changes in the patient health over time, requiring modifications in the care plan. These modifications can require the addition of components not available in deployment time.

Considering a scenario where the care plan is updated with a new health problem,
cardiac insufficiency for example, it is necessary to monitor both the heart rate and the weight of the patient. Figure 22 shows the profile with the context rule that checks the care plan (line 03), and a contract to perform the architecture adaptation.

This contract, like the first one, is associated with the SPL architecture, and has an adaptation service (lines 06-13). Three new components are added and interconnected to other architecture components (lines 07-13), considering that a new health problem is detected in the patient care plan (line 03 of CardiacInsufficiencyProfile). When this service is applied, the product architecture resulting from the adaptation allows monitoring of a patient with cardiac insufficiency, and also keeps available resources for the monitoring related to hypertension. Figure 23 shows the product architecture aiming at the needs of a patient with both hypertension and cardiac insufficiency.

It is important to notice that both contracts continually check the context information corresponding to the patient health problems, using the context rules described in profiles. If, at any time, a context rule cannot be met, or, if a component is not available (e.g., a sensor fails or is not installed at home), the adaptation service is interrupted by the configuration support framework, and the no_service state is reached (line 16 of the Figure 22, and line 19 of the Figure 19). Because of this, part of the product architecture, established by the adaptation service, becomes inactive. For example, if the doctor decides to no longer monitor the patient for hypertension, changing therefore his/her care plan, the product architecture will be adapted keeping only the components related to cardiac insufficiency. Figure 24 shows the resulting architecture.

In addition, new contracts can be created at any time, and associated with the SPL architecture. For example, a contract to deal with specific needs of diabetic patients can be defined, independently of the other contracts as well as of the SPL architecture.
6. Related Work

Several works in SPL discuss static variability management not considering changes that may occur at runtime [van Gurp et al. 2001, Northrop 2002, Krueger 2002]. Nevertheless, others propose variability management at runtime based on parameterization and policies techniques [Goedicke et al. 2004, Svahnberg et al. 2005]. There is also research about SPL feature model such as [Lee and Muthig 2006] which proposes a technique to develop dynamically reconfigurable components, and [Fernandes et al. 2008] which proposes an extension to the feature model to represent contextual information. These works do not adopt an architectural representation, unlike our architecture-centered approach where components and connectors are treated as first class entities.

There are several works that address dynamic adaptation based on architectural models. [Hallsteinsen et al. 2006] consider dynamic variabilities such as configurations, similar to our approach, and the reconfiguration is based on utility functions. Our negotiation mechanism can also be driven by utility functions [Petrucci and Loques 2007]. We also experimented with scripts to select the best configuration in applications with a great number of states, a situation where to write a state machine is difficult.
Figure 21. Product architecture for patient with hypertension.

```
01 profile {
02   CarePlan.patientId = %patientId;
03   CarePlan.healthProblem = "Hypertension";
04 } cardiacInsufficiencyProfile;
```

(a)

```
05 contract cardiacInsufficiencyContract (patientId) {
06   service { ...
07     instantiate PhysiologicalSensor as heartRate
08     with cardiacInsufficiencyProfile;
09     instantiate PhysiologicalSensor as weightScale
10     with cardiacInsufficiencyProfile;
11     instantiate MedicalRules as cardiacInsufficiencyRules
12     with cardiacInsufficiencyProfile;
13     link heartRate to cardiacInsufficiencyRules;
14     link weightScale to cardiacInsufficiencyRules;
15     link cardiacInsufficiencyRules to decisionModule;
16   } cardiacInsufficiencyService
17   negotiation {
18     not cardiacInsufficiencyService -> no_service;
19   };
```

(b)

Figure 22. a) Profile for a patient with cardiac insufficiency; b) Contract for a patient with cardiac insufficiency.

[Morin et al. 2009] consider four aspects related to adaptation: variability, environment and context system, system architecture and adaptation logic. Architectural configurations are built using techniques based on aspect-oriented modeling, whereas our approach provides separation of concerns through other mechanisms, since contracts represent variability in a high level of abstraction and can be described regardless of the architecture.

Rainbow [Garlan et al. 2004] uses invariants coordinated with adaptation strategies, which are similar to profiles and services used in the contract, and adaptations are embedded in procedures that implement the configuration actions. Differently, adaptations in our approach are governed by contract based on an ADL, which allows formal validation of the adaptation procedure before the product deployment [Braga et al. 2009, Beugnard et al. 1999].

The proposal described in [Bencomo et al. 2008] is complementary to our research. Authors distinguish two types of variability: environment variability that defines the conditions under which a system must adapt, and structural variability that defines
the resulting architectural configurations. Reconfiguration is done through a component-based middleware and uses a state machine to represent configurations system.

Regarding the home health care domain, some systems have been proposed (e.g., [ElHelw et al. 2009, Leijdekkers et al. 2007]), however they do not address personalization aspects as argued in Section 3. [Laguna et al. 2009] present a similar idea to the SPL shown in Figure 4 in order to customize systems. However, the authors do not deal with various features addressed by our proposed system, such as the use of a context service [Sztajnberg et al. 2009] and the integration of patient’s physiological and behavioral information aiming at the identification of abnormal conditions of the patient [Copetti et al. 2009].
7. Conclusion

This technical report presented an approach for dynamic variability management in SPL. In our approach, architectural contracts are the key elements to manage dynamic variabilities. These contracts are handled by a configuration support framework responsible for context monitoring and architecture adaptation control. In order to demonstrate the feasibility of the approach were described scenarios involving dynamic variabilities of a SPL for ubiquitous/pervasive home health care systems.

The main contribution of the proposed approach is the possibility of expressing dynamic variabilities at a higher level of abstraction. The fact that contracts are not part of the SPL architecture, but are just associated with it, allows dynamic variabilities to be treated separately from the commonalities, providing different architectural views, as suggested by [Kramer and Magee 1997]. This is beneficial in terms of separation of concerns by facilitating the components reuse among the products that make up the line. Moreover, it facilitates the evolution of both SPL and the contracts, because SPL are not aware of the existence of contracts and additional contracts can be created and incorporated to the SPL during runtime. However, this flexibility demands strict configuration management support to avoid incompatible bindings of SPL components and contract versions.

A contract description language underpins our approach for dynamic SPL development. Specifically, it allows, at the architectural level, the representation of both the adaptation actions (services) taken in the architecture and the context rules (profiles) used to guide such adaptations. Moreover, the semantics of the contracts can facilitate verification procedures and formal validation of its specifications, even before the software product deployment [Braga et al. 2009].

In this technical report we describe our initial results, in the home health care domain, which were focused on the support of SPL applications that have to meet dynamic adaptation requirements. Currently, we are investigating techniques to adapt contracts at runtime, or even create and instantiate new contracts, so they can handle requirements not available in development time, as argued in [Bencomo et al. 2010].

Acknowledgments

The authors would like to thank the financial support provided by CAPES (Coordination for the Improvement of Higher Level Personnel - Brazil), CNPq (The National Council for Scientific and Technological Development - Brazil) and FAPERJ (Rio de Janeiro State Foundation for Research Support - Brazil).

References


