Runtime Variability for Dynamic Reconfiguration in Wireless Sensor Network Product Lines

Óscar Ortiz, Ana Belén García Rafael Capilla Jan Bosch Mike Hinchey

ABSTRACT

Runtime variability is a key technique for the success of Dynamic Software Product Lines (DSPLs), as certain application demand reconfiguration of system features and execution plans at runtime. In this emerging research work we address the problem of dynamic changes in feature models in sensor networks product families, where nodes of the network demand dynamic reconfiguration at post-deployment time.

1. INTRODUCTION

A Software Product Line (SPL) maximizes reuse by exploiting both the common and variable aspects of systems; system features are described in the architecture using a complementary representation model called a feature model [1] [2]. The structural variability, often seen as orthogonal to other software artifacts, shows a hierarchy of a system's features which are bound to concrete values at different binding times [3] [4].

Today, the ever increasing complexity of software systems demands more and more runtime capabilities (e.g., SOA applications) that enable speedy reaction to new context conditions and better support for evolution of products. Many software systems that require adaptation to changing environments (e.g., self-adaptable and self-healing systems [5], [6] autonomic computing [7], [8], ubiquitous systems, robotics, etc.), need the activation/deactivation of system features [9] autonomously. One domain where runtime concerns play an important role is that of Wireless Sensor and Actuator Networks – WSANs). A WSAN encompasses a set of interconnected sensors and actuators that react at runtime to environmental changes and other contextual information. Different sensor networks are of major interest for DSPLs such as we highlight in this work.

This position paper describes a runtime variability mechanism able to modify the structural variability of a WSAN dynamic software product line. More specifically, we address the problem of modifying structural variability at runtime when systems using context information need to change their current configuration, and we use a novel mechanism that helps to modify system variants dynamically. The reminder of this article is organized as follows: In section 2 we describe the major characteristics and challenges of sensor networks, focusing on software issues that can be described using a DSPL. In section 3 we describe a runtime variability approach. Section 4 outlines a motivating example of a sensor network product family that uses a runtime variability mechanism. Section 5 addresses related work and in section 6 we draw conclusions and describe future work.

2. WIRELESS SENSOR AND ACTUATOR NETWORKS FAMILIES

In this section we describe the main characteristics of WSANs and how they can be modeled as a product family. In addition, we focus on those WSAN features that are susceptible to being managed by a DSPL approach. Generally speaking, a WSAN is composed of a potentially large number of small (resource-constrained) nodes — called motes, sensor nodes, or simply sensors — that sense, actuate, process and communicate in order to provide functionality that is not usually obtainable any other way. They are naturally embedded in the real world, and thus one of their main advantages is that they behave in a strongly context-aware manner, sometimes in places where humans or other type of systems cannot be present because of accessibility, scalability or safety reasons [10].

We describe in the following bullet points those issues that become more significant for dynamic reconfiguration in WSANs (for more details see [10], [11], [12] and [13]):

- *Strong energy restrictions.* This is caused by the fact that in many cases the nodes operate on batteries (possibly combined with some form of energy harvesting) and the global network has to survive for a long period of time before it ultimately stops working. Nodes operating on mains do not have this restriction, although there may be a mix of both types of devices in the same network. Frequently, the activity that mostly consumes a node's energy is wireless communication (see, for instance, [14]), and this may lead to the need for dynamic changes in the communication parameters and roles of specific nodes according to the distribution of the energy in the network.

- *Distribution*. WSAN are distributed systems by nature, something that may affect runtime variability in the sense that changing features at runtime may involve modifications in several nodes, often implying some form of multi-hop communication of the new code or modules. As stated in the previous bullet, energy restrictions may call for carefully designed dynamic reconfiguration processes.
- *Unattended (autonomous) operation*. As nodes can be deployed in physically inaccessible or dangerous areas, automatic ways to evolve the network (e.g., adding a new functionality or feature) are needed.
- *Heterogeneity*. In a WSAN the nodes can be different (e.g., different sensors) and exhibit different features (e.g., the logical communication topology may be cluster-based, so that a subset of the nodes will act as cluster heads and have more sophisticated routing tasks). More complex functionality may require more energy consumption that should be placed in less-constrained nodes if possible. Such distribution of functionality can be dynamic because the energy depletion of the nodes may not be equally fast.
- *Software adaptability*. Since WSANs usually have to survive with no (or minimal) changes in deployed hardware for long periods of time, it is possible that dynamic software changes have to be performed during their runtime life. These changes may be triggered by very dynamic occurrences (e.g., a failure in some nodes or a fluctuation in a context parameter that make it advisable to change the network operation) or by less-frequent events such as the conscious decision to add, modify or remove a certain functionality.
- *Self-* properties*. Self-healing, self-management and resilience are examples of non-functional features that are expected from a WSAN. As a typical example of this, when a node fails, the network must be capable of reasonably continuing performing its tasks if possible, which might require dynamic reconfiguration.

2.1 Dynamic reconfiguration in WSANs

Today, a WSAN mixes context-aware properties, autonomic computing, and smart devices to sense environmental conditions and react using a set of actuators. There have been recent approaches that tackle the problem of using context information to engineer a DSPL. Clear examples come from the autonomic computing field, where system features are reconfigured dynamically in the context of a DSPL for autonomic homes and to enhance self-management capabilities as well [15]. Another example of the use of autonomic computing, this time specifically for evolving Wireless Body Sensor Networks*, can be found in [16].

WSANs demand reconfiguration activities at runtime, as previously stated. Moreover, a reconfiguration normally involves changes in specific parts of the code. This is why, in order to save energy, some proposals use a component-based framework that allows the dynamic reconfiguration of nodes by changing, adding or deleting specific components instead of the complete software image ([17], [18]). This reconfiguration activity needs to transmit only a subset of all the application components, which is easier when components are loosely coupled. The aforementioned proposals provide a tool for dynamic reconfiguration, but they do not establish a systematic relationship between the components and the system features. The existence of these proposals provides a basis on which to build our model, which would ease the task of automatically reconfiguring the WSAN in an energy-aware manner (i.e., communicating and changing only the required software components).

Other related work uses feature models to describe WSAN evolvable products (e.g. [19] describes an ambient assisted living case study, and in [15] the authors propose a DSPL representation for autonomic homes). There are also authors that have developed a feature model tool specific for WSANs [20]. In [21], the authors describe middleware services for the dynamic reconfiguration of a WSN. They use feature models extended with context parameters capable of triggering a set of pre-defined reconfiguration plans. However, they do not include the possibility of changing system features at runtime, and all the possible reconfiguration plans have to be previously uploaded into the network nodes.

Our proposal models WSANs as a SPL which includes dynamic features that require runtime changes. Hence DSPL solutions can be used to deal with runtime reconfiguration problems.

2.2 Featuring WSAN Product Families

There are many aspects in a WSAN that may be considered as complete products by themselves, ranging from the hardware nodes, to the radio transceiver or processor microchips, sensors/actuators, operating systems, routing algorithm and protocols, middleware approaches, and up to the application-related functionalities. Some of these may not substantially differ from other cyber-physical systems.

However, in this work we focus on those properties that a whole WSAN product may expose to the designer of a particular application. For instance, a particular WSAN product family may be "indoor efficiency & comfort systems". Specific products of this family may be used to build a smart home application or a smart office application. Other examples of WSAN product families are "outdoor fire detection systems" and "indoor smart environmental and guidance systems". The complete WSAN, seen as a product itself, presents the important functional and non-functional requirements previously described, which make them a challenging area for applying the DSPL concept. We expect that our work will be beneficial for the automation of the reconfiguration of these systems without disregarding any of the important restrictions.

We propose to describe WSAN families using a feature model that encompasses the following main characteristics, (see Figure 1):

- ***Communication***: By definition, any WSAN has to communicate something (measurements to the sink, commands to specific nodes, alarms, etc.). From the point of view of the user (the designer of a specific application), both the communication type (when the communication takes place) and the type of node identities (how we address the source and destination of a piece of information) are highly significant. We identified three features under "Communication type" features. "Periodical" means that the communicated information (e.g., a sensor measurement) is sent with a fixed frequency. This is useful, for instance, to an application that needs the histogram of temperatures in an

* Wireless Body Sensor Networks (sometimes also called "Wireless Body Area Networks") are composed of wearable sensors and devices that are located on a person's body in order, for instance, to monitor and control his/her health and physiological status.

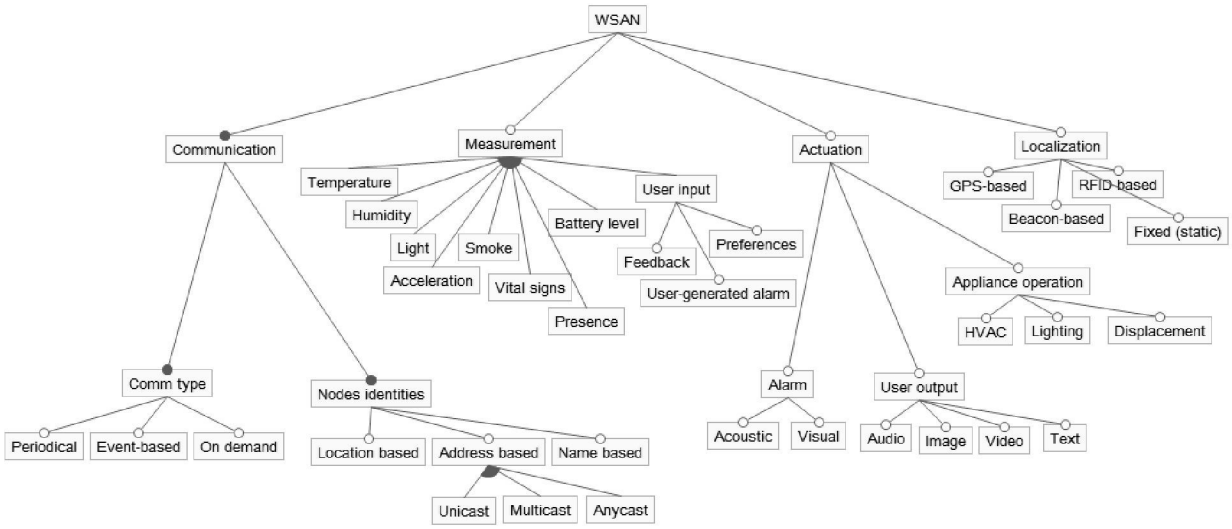

Figure 1. A Wireless Sensor Actuator Network feature model.

area obtained in periodic 5-minute samples. However, if we are only interested in receiving the temperature reading when it exceeds a certain threshold, then an "Event-based" communication type is adequate. "On demand" represents the ability of asking a specific node (or group of nodes) to provide some information, and will usually be triggered by the management system. Under the "Nodes identities" feature we include "Location based" (the ability of sending information to the nodes that comply with some position-related parameter, such as "all nodes inside the North-East portion of an area"), "Address based" (where each node or group of nodes has a numerical address, whether IPv6 or any other type), and "Name based" (encompassing other logical naming of the nodes that may be instrumented by the routing protocol, such as "all the nodes that measure humidity").

- ***Measurement***: Any context information that is of interest to an application has to be sensed using the appropriate sensors. There are myriads of different sensors; we have included in Figure 1 a sample of some usual measurements. We found it relevant to include any form of voluntary user input as a specific case of a "measurement", since this information can be treated basically as an input taken from the WSAN environment. Common examples come from the user stating some personal preferences (such as "only children-adequate content") or the ability for pressing a button to trigger a health alarm. Both "User input" and "Vital signs" may require people to have wearable or portable devices (nodes). It is also noteworthy that the battery level, even if related to the internals of the node, is also considered by us as a measurement that may have to be taken into account to trigger some reconfiguration tasks.

- ***Actuation***: This feature includes any action that modifies one or several physical parameters of the environment. We identify three main types of actuation, as can be seen in Figure 1. (i) "Alarm", whether "Acoustic" or "Visual", is self-explanatory. (ii) With "User output" we mean any type of information offered to people, in a human-understandable format. This may be taken as a reciprocal of the previously mentioned "User input" feature under "Measurement". (iii) Finally, a typical "Appliance operation" example is the operation of the HVAC system in order to enhance the comfort or the safety of people. Another example is "Displacement", a specific form of actuation consisting of mechanically moving a node or device (not to be confused with the externally-sourced mobility that occurs when a person carries a node, for instance).

- ***Localization***: Some applications are location-aware, at least for some of their functionality. There is the possibility of manually establishing the location information in fixed nodes ("Fixed (static)" feature) but also of having radio-based algorithms or devices that provide the location information for a node. "GPS-based" localization is only available outdoors, while "Beacon-based" localization requires some fixed nodes placed in known positions to wirelessly communicate with the nodes we want to locate. "RFID-based" localization relies on the proximity of the target to a specific known fixed location. Especially challenging is the tracking of moving nodes or objects, because the position has to be dynamically updated with enough granularity and frequency.

Each feature of our proposed WSAN feature model is related to a concrete set of software and hardware modules that are in charge of actually implementing the related functionality. For instance, "Temperature" is a property that requires that some hardware modules be present (specifically temperature sensors in at least a subset of the nodes) and some basic software modules be implemented (usually some basic operating system function group that provides access to sensor readings). Other properties may imply not only a more numerous set of hardware and software modules but also some kind of distributed logic. An example of this is the "Multicast" feature (one of the "Communication" sub-tree leaves). In order to have multicast communication, it is necessary that the nodes have the notion of their address pertaining to multicast groups, and equally importantly, a multicast routing protocol has to be in place in a consistent manner in all nodes. As any non-trivial routing protocol, this is a classic example of distributed functionality.

There are also restrictions that apply to the properties of this FODA model. To give an example, the communication-related "Location-based" property (that means the capability of addressing one or a set of nodes by their location in order to send information to them) requires that at least one form of "Localization" is activated in the corresponding WSAN product.

One example of a run-time dynamically reconfigurable variant is the addition of a novel type of measurement provoked by an upgrade or addition of nodes with a new type of sensor, which may require the deployment, activation, and configuration for that sensor. Another example is the activation of a communication property;, e.g., "Event-based". In this case, it is most likely necessary to deploy, configure and activate a new module in all the nodes that in a distributed manner implements a publish-subscribe-like interaction model. This module will be also related and interacting inside each node with pre-existing modules such as those in charge of certain sensor readings.

3. A RUNTIME VARIABILITY MODEL FOR DSPLs

In this section we describe which runtime changes at the variant level we manage in order to support the dynamic reconfiguration of the structural variability of a WSAN. Dealing with open variability models where variants can be added, changed, or removed is not easy, as it is impossible to foresee all unexpected variations in a system. Therefore unexpected variability must be supported in a controlled way to predict the evolution of the DSPL under certain conditions (e.g.: a DSPL provides a runtime variability mechanism which allows, for instance, the inclusion of variants that satisfy certain requirements, or activate and deactivate features during system execution in order to react to different context conditions). Predicting variability in a controlled way helps to anticipate that certain changes in the variability model can be supported without a complete redesign and re-implementation of the product features.

As the modification of variation points requires more human intervention in most cases, and it can be automated only in limited situations, we will focus on the modification of variants at runtime. We are aware that changing variation points is also a design problem, as by now it cannot be fully automated, but we do not focus in this work on how much automation a SPL needs to become a DSPL. Hence we support the following three scenarios:

- *Adding a variant* implies that we need to define in which place of the feature model the variant will be added, and if it will belong to an existing variation point. The logical formulas in the feature model must be modified accordingly to the new variant. In addition, we need to check the existing constraint rules when the new variant is added (e.g., mainly require and exclude rules) to detect potential incompatibilities. In addition, we should know if the type of the new variant (e.g.: string, numerical, Boolean), often used to define the allowed values, is compatible with the types of related variants (e.g., the new variant is added to an existing variation point).
- *Removing a variant* means that the variant will be dropped from the logical formula that connects the variant in the feature model. Also, constraint rules must be revisited when removing a variant.
- *Changing a variant* may imply in its simplest form just changing their values, while in a more complex situation it could mean moving the variant to a different location in the feature model. In this case, moving variants from one place to another, we can implement such operation as a removal task followed by an addition task of the variant.

3.1 Super-types Enabling Runtime Variability Changes

In order to support the aforementioned scenarios, we introduce the notion of a Super-type (ST). Using super-types in system features we capture the essence of variation points as we can group variants under the same or a compatible super-type or specific category. Super-types encompass the basic types (i.e., Boolean, string, integer) commonly used by system features and enable a superseding category for those features that share some common functionality (e.g., features describing ambient measurements may be labeled with an "Ambient" super-type). Therefore, we define and use the notion of supertype with the following purpose:

- Provide a superseding taxonomy over the aforementioned basic types, where system features defined in a variability model can be assigned to one or several super-types.
- Each DSPL for a given application domain can define its own super-types (e.g., multimedia, security) to allocate specific system functionality.
- Our use of the super-types to change the variants can be dynamic, as we can compare the super-type of a new or existing feature to add, remove, or replace a feature which has the same or a compatible super-type.

While basic types of variants define the type of allowed values for a specific system option, super-types group variants that encompass a related functionality. In addition, super-types can be used to filter out and to visualize only a subset of the feature model, in particular when the number of variants exceeds the visualization capabilities of the screen. In our model, we define a variant enabled with super-types as:

$$\text{Variant } (V_i) = ([Va_1 \ldots Va_n], T, [ST_1 \ldots ST_n])$$

where each variant (V_i) representing a system feature has several allowed values (Va_i) that can be organized in different ways (e.g.: ranges, lists), poses a type (T), and belongs to one or several compatible super-types (ST). In section 6 we show an example where a subset of WSAN features are organized into one or several super-types.

3.2 Prototype Software

As a proof of concept, we introduced the notion of super-types into an existing prototype tool called VMWT (Variability Modeling Web Tool [22]) developed at the Rey Juan Carlos University in 2007. We implemented a specific module called Alter Product Runtime Mechanism (APRM), which simulates the runtime changes performed over a feature model. Basically, the APRM module checks the super-type of the variants when a runtime change is needed and modifies the structural variability. To achieve this, the APRM builds a parallel environment where the changes are applied in a secured form without affecting the main system during its execution, and uses replicated data without affecting its normal functioning. After all changes have been applied, the APRM incorporates the new data into the real environment. The APRM module stores in a database the features and their relationships and it uses an array to build and dynamically re-build the changes performed over the variants. Because new features may fit several super-types at the same time, we need policies to know the exact point where these

features will be anchored. As a WSAN could add a new node to the network with features supporting new functionality, variants can be inserted dynamically if they possess the same or a compatible super-type with the existing features; as in other cases a new super-type must be defined before including the variant.

Removal operations of variants are easier because we only need to run automatic scripts to check potential inconsistencies in the dependency rules once we drop the variant in the feature model. Changing the values of variants can be simply done by uploading dynamically a configuration file, while moving a variant to a different location is performed as a removal operation followed by an adding operation.

Moreover, adding new variants may introduce new requires, excludes, and operational dependencies (closer to runtime concerns [9]) between the new variant and the existing ones. This task can be done automatically by uploading dynamic configuration files containing new rules, but we need to run scripts for checking and detecting potential inconsistencies in the new feature model. To leverage the degree of automation when we prove the correctness of the new feature model, DSPL designers can pre-check the dependency rules that will be uploaded automatically before variants are modified. In the APRM tool we can simulate the inclusion of rules that are introduced manually by an operator, and see the effects over an existing or modified feature model.

4. MOTIVATING EXAMPLE

As a motivating example to preliminarily test our proposal, we describe in this section a case for an "indoor smart environmental and guidance systems" product family (Figure 2) derived from the feature model shown in Figure 1. We also show in Figure 2 those active features for a specific WSAN museum guidance product. The summary of the high-level features included in the FODA related to the "indoor smart environmental and guidance systems" product family is as follows:

- *Communication*: With the exception of "Name-based" and "Anycast", all the other features from the general WSAN FODA may be useful in products of this family.
- *Measurement*: Ambient measurements are of potential use for products of this family, as are user preferences (both may affect the guidance advice to be provided to the users). Presence detection may be also interesting.
- *Actuation*: Alarms, Appliance operation (basically for enhancing comfort) and User output (for providing the guidance advice to human users) are significant in this family.
- *Localization*: This product family requires the localization of some mobile nodes, which will be probably carried by people, and the presence of fixed nodes that may act as beacons and have fixed known positions.

As shown in Figure 2, the specific museum guidance product has some of its features active while others remain deactivated. For example, guidance instructions are only given in audio form, and ambient measurements are sensed and communicated in a periodic manner. In Table 1 we show an example where a subset of the WSAN product family features are organized into one or several super-types. Later we exemplify how this definition may be useful for runtime variability.

All ambient measurements (Temperature, Smoke, Humidity, and Light) belong to the "Ambient" ST. Besides, both "Temperature" and "Smoke" may be used to raise alarms (e.g., a measurement with an abnormally high value may be the symptom of a fire), and consequently they also belong to the "Security" ST. Regarding the dynamic reconfiguration capabilities of our WSAN we provide the following two scenarios of use:

Scenario 1: The first example is an upgrade of the museum guidance product that consists of immediately triggering events when any ambient measurement exceeds configurable thresholds (so that we do not have to wait until the next period to know about a potentially dangerous condition). This requires the activation of the (currently inactive) "Event-based" variant, which should eventually cause the upgrading (or activation) of specific software modules inside the nodes to implement a publish-subscribe communication mode. Also, the properties of the features

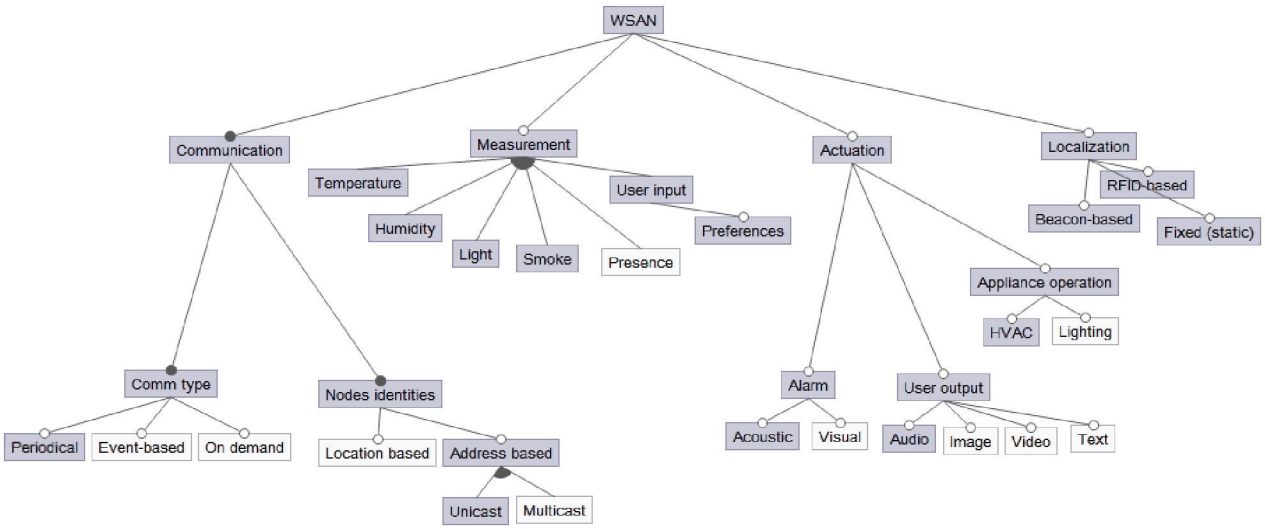

Figure 2. "Indoor smart environmental and guidance systems" WSAN product family. Darker blue boxes are the active features for the Museum Guidance product.

pertaining to the "Ambient" ST have to be modified and the appropriate thresholds have to be set. Again, this will cause some appropriate updates to the software at the nodes. Here we are assuming that the features contain properties or attributes that are subject to runtime variability, being the threshold values part of these possible attributes. Figure 3 contains a pseudo-code representation of this runtime variant modification.

```
{Activate_Feature(Comm_type.Event-based)} AND
{For all Features in "Ambient" ST: Configure threshold
values}
```

Figure 3. Events trigger for ambient measurements

Table 1. Example of super-types in a WSAN product family

Feature(s)	Type (T)	Super-types (STs)
Temperature, Smoke	Range of numerical values	Ambient, Security
Humidity, Light	Range of numerical values	Ambient

Scenario 2: The second example is related to the addition of new hardware: pollen sensors, useful for warning people with allergies. In this case a new variant has to be added to the feature model, called "Pollen", which should be placed under "Measurement" in the FODA tree shown in Figure 2. This new feature will be assigned to the "Ambient" ST, something that will automatically allow for its placement in the correct variation point of the tree and cause the configuration of the corresponding thresholds (if the upgrade from the previous example is already in place).

In order to support the aforementioned scenarios, once we derive the WSAN feature model of Figure 2, we simulated the runtime changes in the feature model using the APRM tool for scenario 2 to add a new variant or system feature by checking its super-type.

5. RELATED WORK

Dynamic Software Product Lines (DSPLs) constitute an emerging approach that uses runtime variability mechanisms [23] to deal with the dynamicity of changing features and dynamic reconfiguration requests in order to adapt systems to varying conditions. In Dynamic Software Product Lines, family members often need to evolve or be reconfigured after deployment. Hence, in the era of post-deployment, many runtime concerns must be addressed at the customer side (e.g., mobile software feature reconfiguration) or autonomously by systems that need to adapt themselves to new requirements and context conditions (e.g., robots). Therefore, runtime architectures must support the reconfiguration of system features and, if needed, modify the structural variability dynamically. There are some recent experiences with the use of DSPLs in various application domains such as SOA (e.g.: [24] [25] [26] [27]) while others exploit context-awareness properties to manage the information at runtime. All these approaches rely on runtime architectures able to manage the dynamicity of changing system features at execution time.

In traditional SPL approaches, software variability (i.e., the structural variability of a set of related systems) was modeled and used statically by the product line process. With the advent of dynamic software product lines, variability is managed dynamically at runtime, and runtime models attempt to manage and change variants during system execution. Some of these changes refer to the activation and deactivation of system features while more sophisticated approaches can add, remove, or modify variants in the feature model. In [26] the authors state many of the challenges concerning the introduction of runtime variability mechanisms that are used by DSPLs. Automating and validating runtime variability is hard,, and open variability models must be flexible enough to support better the evolution of software products and post-deployment changes, including runtime binding and rebinding tasks. To date, few experiences have dealt with changing variants at runtime [28] [29] [30] [31] [32]. The modification of variants at runtime often requires some kind of meta-information or reflective technology to support the inclusion and removal of variants automatically or semi-automatically, but changing variation points dynamically is not supported by current approaches as human intervention is needed to decide on the logical formulas connecting their variants. Other attempts use compositional approaches to support feature reconfiguration at runtime [33]. Also, in [34], three different strategies rely on the Common Variability Language (CVL) [35] are used for implementing variability transformations and synthesis in smart-home systems. The CVL language includes constructs to compose three types of substitutions according to the type of alternatives able to perform reconfiguration activities of the base model. Rebinding and reconfiguring variability models with moderate effort is, therefore, a major issue, and tools to evaluate the effect of dynamic reconfigurations and the changes in the state of features are needed [36].

Finally, the notion of *context variability* shares some similarities with our work. In [37] the authors combine goals models with contextual information for product line derivation. They state that context can be a main factor for determining the products to derive, and context information can be incorporated in variability models. In [38] context variability is used for multiple product lines, and the authors suggest a context variability model associated to an existing feature model to constraint the variability model. Contexts are used as general classifiers in which the product is used. Our approach uses the super-types as a taxonomy of features, but we do not duplicate the existing feature model and we do not introduce new relationships, as in [38]. Also, context features are used differently and the authors in [38] do not suggest the automation of changes in the feature model, as they focus on product derivation rather than on changing the structural variability dynamically.

6. CONCLUSIONS AND FUTURE WORK

One of the main conclusions we can draw from this emerging work is that system families using context-aware information that require runtime changes are quite suitable to be described using a DSPL. Like other related systems, WSAN families can be described using features models but highlight those runtime characteristics that require the support of dynamic variability.

The proposed super-type-enabled runtime variability mechanism extends previous feature models and provides a simple way to manage and change variants dynamically using the notion of super-types. These super-types enable a classification of high-level system properties and functionality that can be used to filter out and select only the specific functionalities that require to be changed dynamically. Hence, variants modified at execution time offer a flexible solution for WSANs that change their nodes or current configuration at runtime. One aspect that may hamper the full automation of changing variants at runtime is the need to check on-line mode constraints and dependency rules in the feature model when, for instance, a new variant or super-type is added dynamically. The APRM prototype simulates runtime

changes in the structural variability where system features can be bound at runtime and post-deployment time in a DSPL.

For future work, we are working on new prototype software that integrates the APRM module into an existing system in order to automate the runtime changes required by WSANs. In addition, we plan to include the reconfiguration functionality described in scenario 1, as this is only supported by the APRM software in a limited way. Finally, more theoretical work has to be done regarding the proposed approach, and especially to provide a meta-model linking the notion of super-type with other typical elements of DSPLs and variability models (e.g., features). A software architecture showing all these concepts together is planned.

7. ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness (Ministerio de Economía y Competitividad), in the framework of the project "Accessible wearable device platform for smart environments" (Ref. TEC2011-28397) and in part by Science Foundation Ireland grant 10/CE/I1855 to Lero—the Irish Software Engineering Research Centre (www.lero.ie). We also thank Alejandro Sánchez for his work in the development of the APRM tool in the Rey Juan Carlos University. Óscar Ortiz and Ana Belén García are members of the R&D centre "Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM)" and of the Department of Telematic Engineering and Architectures (Departamento de Ingeniería y Arquitecturas Telemáticas – DIATEL, UPM).

8. REFERENCES

[1] K. Pohl, G. Böckle, F. van der Linden, *Software Product Line Engineering Foundations, Principles, and Techniques.* Springer Verlag, 2005.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-90-TR-21, 1990.

[3] A. van der Hoek, "Design-time product line architectures for anytime variability," Science of Computer Programming. Special Issue on Software Variability Management, vol. 53, no. 3, pp. 285–304, 2004.

[4] C. Elsner, G. Botterweck, D. Lohmann, W. Schröder-Prekshat, Variability in Time - Product Line Variability and Evolution Revisited. 4th International Workshop on Modelling Variability of Software-intensive Systems (VaMoS 2010), Essen, Germany, 131-137, 2010.

[5] D. Garlan, S.W., Cheng, A.C., Huang, B. R., Schemerl, P. Steenkiste, Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. IEEE Computer 37(10), 46-54, 2004.

[6] J.C. Georgas, A. van der Hoek, R.N. taylor, Using Architectural Models to Manage and Visualize Runtime Adaptation, IEEE Computer 42(10), 52-60, 2009.

[7] J. O. Kephart, D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[8] C. Cetina, Pau Giner, J. Fons, V. Pelechano, Autonomic Computing Through Reuse of Variability Models at Runtime: The Case of Smart Homes, IEEE Computer 42(10), 37-43, 2009.

[9] J. Lee, and K. Kang, Feature Dependency Analysis for Product Line Component Design, *International Conference on Software Reuse*, LNCS 3107 Springer-Verlag, pp. 69-85, 2004.

[10] I. F. Akyildiz, et al. "Wireless Sensor Networks: A Survey." *Computer Networks* 38 (2002): 393-422.

[11] M. Turon. "MOTE-VIEW: A Sensor Network Monitoring and Management Tool". *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors.* Washington, DC, USA: IEEE Computer Society, 2005. 11-17.

[12] European Research Cluster on the Internet of Things. "Internet of Things Strategic Research Roadmap". 2011. Available online: http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2011.pdf

[13] G. Karsai, et al. "Evolving Embedded Systems." *Computer* 43.5 (2010): 34-40.

[14] A. Bachir, M. Dohler, T. Watteyne, K.K. Leung. "MAC Essentials for Wireless Sensor Networks". IEEE Communications Surveys & Tutorials. Volume: 12, Issue: 2. 2010. Pp. 222 – 248.

[15] C. Cetina, P. Trinidad, V. Pelechano. Mass Customization along Lifecycle of Autonomic Homes. In: Proceedings of 3rd Dynamic Software Product Lines (DSPL), Limerick, Ireland, 2009.

[16] G. Fortino, S. Galzarano, and A. Liotta. "An Autonomic Plane for Wireless Body Sensor Networks". *Computing, Networking and Communications (ICNC), 2012 International Conference on.* 2012. 94-98.

[17] D. Hughes, et al. "LooCI: A Loosely-Coupled Component Infrastructure for Networked Embedded Systems". *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia.* Kuala Lumpur, Malaysia. New York, NY, USA: ACM , 2009. 195-203.

[18] A. Taherkordi, et al. "Optimizing Sensor Network Reprogramming Via in-Situ Reconfigurable Components." *ACM Transactions on Sensor Networks* 9.2 (2013): 1-37. Available online: http://hal.inria.fr/docs/00/65/87/48/PDF/RemoWare_TOSN.pdf.

[19] T. Patzke, L. Vajda, A. Török. Evolving heterogeneous wireless sensor networks - an assisted living case study. Varga, A.K.: Regional Conference on Embedded and Ambient Systems, RCEAS 2007. Proceedings. Selected Papers: 22-24 November 2007, Budapest. Budapest, 2008, pp. 89-93.

[20] P. Boonma and J. Suzuki. Model-driven performance engineering for wireless sensor networks with feature modeling and event calculus. In Proceedings of the 3rd workshop on Biologically inspired algorithms for distributed systems (BADS '11). ACM, New York, NY, USA, 17-24, 2011.

[21] N. Gamez, L. Fuentes, and M. Aragüez. *Autonomic Computing Driven by Feature Models and Architecture in FamiWare*. Eds. Ivica Crnkovic, Volker Gruhn, and Matthias Book. 6903 Vol. Springer Berlin / Heidelberg, 2011. Lecture Notes in Computer Science.

[22] R. Capilla, A. Sánchez, J.C. Dueñas, An Analysis of Variability Modeling and Management Tools for Product

Line Development. In proceedings of the Software and Services Variability Management Workshop: Concepts, Techniques, and Tools. Helsinki University of Technology, Helsinki, Finland, April 19-20, 32-47, 2007.

[23] S. Hallsteinsen, M. Hinchey, S. Park and K. Schmid, Dynamic Software Product Lines, IEEE Computer 41(4), 93-95, 2008.

[24] S. Hallsteinsen, S. Jiang, and R. Sanders, Dynamic Software Product Lines in Service Oriented Computing, In: Proceedings of 3rd Dynamic Software Product Lines (DSPL), Limerick, Ireland, 2009.

[25] H. Gommaa, K. Hashimoto, Dynamic Software Adaptation for Service-Oriented Product Lines, 15th International Software Product Line Conference (SPLC), ACM DL, 2011.

[26] P. Istoan, G. Nain, G. Perrouin, J-M. Jézéquel. Dynamic Software Product Lines for Service-based Systems. 9th International Conference on Computer and Information Technology, 193-198, ACM DL, 2009.

[27] H. Shokry, M. Ali Babar. Dynamic Software Product Line Architectures Using Service-based Computing for Automotive Systems. 2nd International Workshop on Dynamic Software Product Lines (SPLC-DSPL 2008), 53-58, 2008.

[28] R. Capilla, J. Bosch. The Promise and Challenge of Runtime Variability. IEEE Computer 44(12), 93-95, 2011.

[29] M. Goedicke, C. Köllmann, U. Zdun, Designing Runtime Variation Points in Product Line Architectures: three cases. Science of Computer Programming 53(3), 353-380, 2004.

[30] A. Helleboogh, D. Weyns, K. Schmid, T. Holvoet, K. Scheltfhout, W. van Betsbrugge, Adding Variants on-the-fly: Modeling Meta-Variability in Dynamic Software Product Lines. Proceedings of 3rd International Workshop on Dynamic Software Product Lines (DSPL 2009), San Francisco, California, USA, 2009.

[31] N. Bencomo, G. Blair, C. Flores, P., Sawyer, Reflective Component-based Technologies to Support Dynamic Variability, In 2nd International Workshop on Variability Modelling of Software-intensive Systems (VAMOS 2008), Essen, Germany, 141-150, 2008.

[32] R. Froschauer, A. Zoitl, P. Grünbacher, Development and Adaptation of IEC 61499 Automation and Control Applications with Runtime Variability Models, 7th IEEE International Conference on Industrial Informatics (INDIN 2009), 905-901, 2009.

[33] F. Damiani, I. Schaefer. Dynamic Delta-Oriented Programming. 5th International Workshop on Dynamic Software Product Lines (DSPL), 2011.

[34] C. Cetina, Ø. Haugen, X. Zhang, F. Fleurey, V. Pelechano, Strategies for variability transformation at run-time. SPLC 2009, San Francisco, California, USA, ACM Proceedings Series 446, 61-70, 2009.

[35] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. K. Olsen, and A. Svendsen. Adding standardized variability to domain specific languages. In B. Geppert and K. Pohl, editors, Software Product Lines, 12th International Conference, SPLC 2008, Proceedings, volume 1, pages 139–148, Limerick, Ireland, 2008.

[36] C. Cetina, P. Giner, J. Fons, V. Pelechano, Designing and Prototyping Dynamic Software Product Lines: Techniques and Guidelines. SPLC 2010, Springer-Verlag LNCS 6287, 331-345, 2010.

[37] R. Ali, R. ChitChyan, P. Giorgini, Context for Goal-level Product Line Derivation, Procedings of 3rd International Workshop on Dynamic Software Product Lines (DSPL 2009), San Francisco, California, USA, 2009.

[38] H. Hartmann, T. Trew, Using Feature Diagrams with Context Variability to model Multiple Product Lines for Software Supply Chains, 12th International Software Product Line Conference, 12-21, 2008.