Context Aware Reconfiguration in Software Product Lines

Jacopo Mauro
University of Oslo
jacopom@ifi.uio.no

Michael Nieke
Technische Universität Braunschweig
m.nieke@tu-braunschweig.de

Christoph Seidl
Technische Universität Braunschweig
c.seidl@tu-braunschweig.de

Ingrid Chieh Yu
University of Oslo
ingridcy@ifi.uio.no

ABSTRACT

Software Product Lines (SPLs) are a mechanism for large-scale reuse where families of related software systems are represented in terms of commonalities and variabilities, e.g., using Feature Models (FMs). While FMs define all possible configurations of the SPL, when considering dynamic SPLs not every possible configuration may be valid in all possible contexts. Unfortunately, common FMs can not capture this context dependence. In this paper, we remedy this problem by extending attributed FMs with Validity Formulas (VFs) that constrain the selection of a particular feature to a specific context and that are located directly within the FM. We provide a reconfiguration engine that checks if the active configuration is valid in the current context and, if not, computes how to reconfigure it. Furthermore, we present our implementation and demonstrate its feasibility within a case study derived from scenarios of our industry partner in the automotive domain.

CCS Concepts

•Software and its engineering → Software product lines;

Keywords

Feature Models, Variability, Software Product Lines, Context

1. INTRODUCTION

SPLs are an approach for structured large-scale reuse where families of related software systems are modeled in terms of commonalities and variabilities [22]. The set of all possible configurations is represented by a *variability model*, such as a FM [16]. An FM structures features along a decomposition hierarchy where individual features represent configurable units of functionality, which may be enriched by additional *feature attributes*–variables with a type defined within the context of a feature. A *configuration* is a valid subset of features along with concrete values for the attributes of all selected features (see Section 2).

In the standard SPL approach, features are selected based on human desires only. Nowadays, however, this may not be enough: with the diffusion of the Internet of Things [5], a very large number of devices have to interact and adapt based on their surrounding. As a consequence, contextual information needs to also be considered in the configuration process. Usually, the development of such systems is performed by defining a context model and rules to specify which configurations and parameters to use for every interesting context [6, 10, 13]. For example, considering the automotive industry, cars may change their navigation software according to their position. If a car is in Russia, instead of using as navigational system the standard GPS, it may use the space-based satellite navigation system GLONASS. Additionally, a cruise control should have a parameter, limiting the maximum selectable value based on the speed limit of the respective country.

Clearly, for systems having a very large number of context or features, the number of rules to write may increase drastically, which makes it very difficult (i) to list all the rules and (ii) to ensure that a valid configuration exists for every possible context. To mitigate this problem, different context extensions have been proposed within the SPL field [2, 11, 14, 20]. Usually these approaches represent the contextual information with additional data structures (e.g., trees, FM, ontologies) and relate the context with the features using external or cross-tree constraints. On the one hand, this allows a solution that is flexible, modular, and expressive. On the other hand, these approaches are not maintainable for medium/big SPLs as they require developers to have a simultaneous global overview and deep knowledge of (i) the FM, (ii) the complex model of the context, and (iii) of the interactions between the FM and context.

In this work, we first address this problem by proposing a framework where contextual information is captured directly within the FM. Our approach is to enrich every feature with an additional formula, called *VF*, that constrains the selection of the feature w.r.t. the contextual values. Locating and presenting this kind of contextual constraint directly on the FM should ease the understanding of the interactions between the contextual change and the features. Hence, to maintain the FM, the developer can focus on its interesting

VaMoS '16, January 27-29, 2016, Salvador, Brazil

DOI: http://dx.doi.org/10.1145/2866614.2866620

subparts without needing a global overview.

Furthermore, in addition to a framework for expressing context-aware FMs, we also address the problem of finding an appropriate new configuration when the current active configuration is invalidated by a context change. As a main contribution, we present HyVarRec–a *hybrid variability reconfiguration engine* that is able to verify if a given configuration is valid w.r.t. an FM, and its context and, if not, proposes the most similar valid configuration for the context. HyVarRec is able not only to select the features needed to obtain a valid configuration but to also automatically set the values for the relevant attributes to make the configuration valid. HyVarRec is the first component of an extensive software reconfiguration framework we are developing for the automotive sector (see Section 3), which we used as motivation for this work.

This paper is structured as follows: In Section 2, we explain the background information needed for our methodology. In Section 3, we present our main motivation: create a reconfiguration framework for the automotive sector. Additionally, we introduce the FM that will be used as running example. In Section 4, we provide the concept of our methodology to model context information correlated with the FM. In Section 5, we introduce and describe HyVarRec. In Section 6, we show the applicability of our methodology with a case study. In Section 7, we discuss related work. In Section 8, we discuss consequences of introducing context within the FM. Finally, in Section 9, we present concluding remarks and a brief outlook on future work.

2. BACKGROUND

In this section, we introduce concepts and technologies our methodology is based on.

Feature Model

To express the variability of a Software Product Line (SPL), we use Feature Models (FMs) [16]. They are structured hierarchically and are usually visualized as a tree. A feature model has a root feature under which the remaining feature model is structured. Thus, except for the root feature, each feature has a parent. The selection of a feature requires the selection of its parent. Additionally, features are organized in *and*, *or* and *alternative* groups. In *and* groups, features can be *optional* or *mandatory*. In *or* groups, at least one feature of that group has to be selected. In *alternative* groups, exactly one feature of that group has to be selected.

To provide additional expressiveness to the variability of features, FMs may be enriched with *feature attributes* [8] consisting of a name and a type. A feature can have an arbitrary number of attributes. The possible values of each attribute are defined by a type and may be further constrained by setting a maximum and a minimum value for Integer attributes and selecting specific enumeration literals for enumeration type attributes, respectively.

Cross-tree constraints on the feature model define dependencies between features, which are not defined by the structure of the feature model. For example, a feature `A` may require a feature `B` but must not be combined with feature `C`. Such constraints can be specified via Boolean formulas. To support constraints on feature attributes, these formulas can be extended by Boolean expressions on feature attributes, e.g., limiting the range of an Integer attribute.

A *configuration* of an FM is a set of selected features and concrete values assigned to the attributes of the selected features. A configuration is *valid* if it complies with the structure of the FM, the value domains of the attributes and all cross-tree constraints of the FM.

Constraint Programming

Constraint Programming (CP) has been identified as the most promising approach for identifying FM errors [7]. A Constraint Satisfaction Problem (CSP) consists of a set of variables, each of which is associated with a finite domain of values that it can take and a set of constraints defining all the admissible assignments of values to variables [17]. When dealing with CSPs, the usual goal is to find one possible solution, i.e., a variable assignment satisfying all the constraints of the problem, by using a suitable constraint solver. A Constraint Optimization Problem (COP) is instead an extension of a CSP where constraints are used to narrow the space of admissible solutions. In this case, the goal is to not just find any solution but a solution that minimizes (or maximizes) a specific objective function.

CSPs and COPs can be defined in different languages and are supported by many tools [4, 9, 12, 24]. In this work, we use *MiniSearch* [25]–a uniform language for implementing constraint programming meta-search that does not place any burden on the author of a constraint solver. MiniSearch uses *MiniZinc* [21], to this day one of the most widely used and supported languages to define CSPs and COPs.

3. MOTIVATION AND RUNNING EXAMPLE

The original motivation for this work is the development of a framework for continuous and highly individualized evolution of distributed software applications focusing on the automotive sector, which can be integrated into existing software development processes.[1]

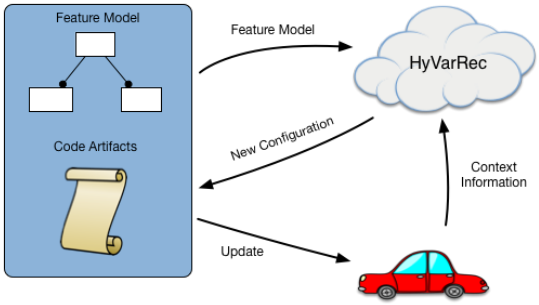

Figure 1: Overview of the reconfiguration framework illustrating the role of the tool HyVarRec

Figure 1 illustrates the HyVar approach with a focus of the contribution presented in this work. The framework relies on a Software Product Line (SPL), which is graphically represented in the left rectangle. The SPL is used to denote the software variants that may be deployed on the remote devices (in the figure, a car is used as an example of a remote device).

[1]This is the main goal of the HyVar project that support this work.

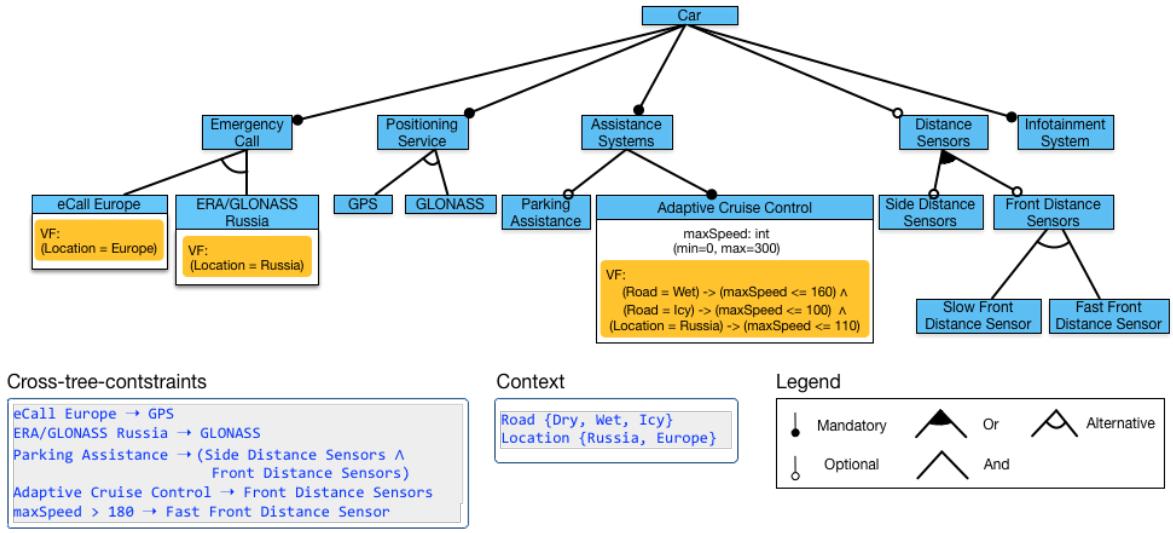

Figure 2: Feature Model for the car SPL of the running example

The FM is represented on the top of the rectangle while the bottom part represents the concrete code artifacts that have to be assembled to produce the actual software variants. The product line declaration incorporates a variability model and provides the connection of the code artifacts with the product features.

The model is linked to a repository (the cloud in Figure 1) that collects the profiles of the remote devices with their context information. These profiles are continuously maintained by the infrastructure which ensures the timely collection of relevant data from the remote devices. The decision to deploy a new configuration on a remote device is triggered by the inspection of the device profile.

In this work, we focus on the core part of the system: how the FM can be enriched by contextual information and on the reconfiguration engine, HyVarRec, that can compute the new configurations. In particular, we abstract from implementation details such as collection of context data, generation of the new variants and the installation of variants on the remote devices.

To demonstrate the concepts of this paper, we introduce a running example. For simplicity, we present a shortened FM of an SPL based on a scenario of one of our industrial partners. Figure 2 shows an excerpt from the FM describing the configuration options of a car.

The car supports different emergency call systems–one for Europe, the `eCall Europe`, and one for Russia, `ERA/GLONASS Russia`. GPS is used as positioning service in Europe and, therefore, the feature `eCall Europe` requires the `GPS` feature. Accordingly, GLONASS is used as positioning service in Russia and, therefore, the feature `ERA/GLONASS Russia` requires the feature `GLONASS`. Among the remaining features, the most important ones are the following: The feature `Adaptive Cruise Control` provides the assistance system. It has an attribute `maxSpeed`, which defines the maximum speed settable for cruise control. Moreover, the value of `maxSpeed` influences the selection of an appropriate subfeature of the feature `Front Distance Sensors`.

To model the impact of the environment on the software system of the car, we provide two contexts: `Road`, which represents the state of the road, and `Location`, which represents the current position of the car.

4. SOFTWARE PRODUCT LINES WITH CONTEXTUAL INFORMATION

Environmental changes have influence on software systems. Thus, it it necessary to capture this influence within the models of the software system to be able to handle these situations. For SPLs, the environment may change the possibility to select specific features. To this end, it is necessary to (i) create a model to represent contextual information and (ii) to model the impact of contextual information on possible feature selection as well as to correlate the contextual information with the FM.

As we explicitly aim at creating a methodology that enables SPL developers to directly see the impact of environmental changes on the features of the FM, we create a model of contextual information, which is focuses on the most essential information. From the use cases of our industry partner, we identified three types of relevant contextual information, which have to be modeled as first-class entities: numerical information, Boolean information, and information with pre-defined values, which can be represented by enumerations. An example for the enumeration type information is depicted in Figure 2, which contains contextual information on the state of the road and the location of the car. For a feature to be selectable under a given context, all of the conditions on that context have to be satisfied. To be able to process the contextual information with a CSP solver, we limit the numerical information to Integer values with a defined domain. The domain of the enumeration type values is limited with pre-defined enumeration literals. Each contextual information can have a current value and the types of the values have to match the type of the respective contextual information and their domain.

To model the impact of contextual information on the feature selection, we introduce the concept of Validity Formulas (VFs) of features. A feature may only be selected if

all its VFs are evaluated to true. A VF is a propositional formula that defines the conditions under which a feature may be selected. However, each VF must define a value for a contextual formula. For example, in Figure 2, for the feature `eCall Europe`, the annotated VF is $Location = Europe$, which implies that `eCall Europe` may only be selected if the car is located in Europe. Moreover, it is also possible to reference other features or attributes in the VF and to define more complex formulas as it can be seen in Figure 2 for the feature `Adaptive Cruise Control`. There, the VF considers two different types of contextual information as well as the value domain of the attribute `maxSpeed`. Thus, it is also possible to constrain the value of attributes with VFs. In general, with a VF, it is possible to create arbitrary propositional formulas with references on contextual information, other features and feature attributes. Each VF is associated with a feature and can be annotated in the FM, as it can be seen in Figure 2. To correlate VFs with features, it is therefore only necessary to relate a VF with its respective feature.

To provide a suitable input format for the reconfiguration engine, it is necessary to translate the feature model and its VFs into constraints. We perform this procedure fully automatically translating the structure of the feature model into constraints as presented by Benavides et. al [8]. For the VFs, small adaptations are necessary to relate the formulas to the features. Thus, as an example, a VF "$Location = Europe$" for the feature `eCall Europe` needs to be translated into the constraint "$eCallEurope \rightarrow Location = Europe$" to ensure that the VF has to evaluate to true if the feature is selected. Accordingly, VFs limiting the domain of attributes need to be adapted to correlate it to the respective feature. Thus, the formula "$Road = Wet \rightarrow maxSpeed \leq 160$" for the attribute `maxSpeed` of `Adaptive Cruise Control` is translated to "$Adaptive\ Cruise\ Control \rightarrow (Road = Wet \rightarrow maxSpeed \leq 160)$".

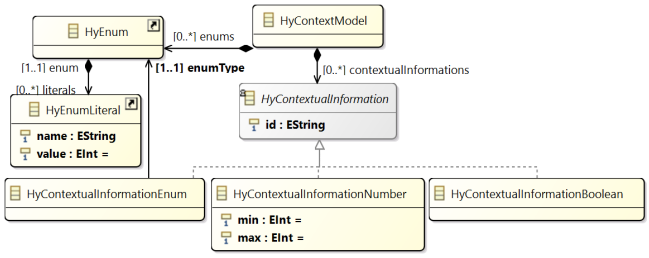

Figure 3: Meta model for Contextual Information

To create contextual information, we provide a meta model using the Eclipse Modeling Framework (EMF)[2], which focuses on the essential information to prevent unnecessary overhead. The meta model of a `HyContextModel` can be seen in Figure 3. Using this model, we can create three types of contextual information: Integer types, Boolean types and Enumeration types. The domain of the Integer types is defined by a minimum and maximum value. For the Enumeration types, a set of valid literals has to be provided, which define the respective domain. Each contextual information can have a current value. The type of the values have to match the type of the respective contextual information and their domain.

[2]https://eclipse.org/modeling/emf/

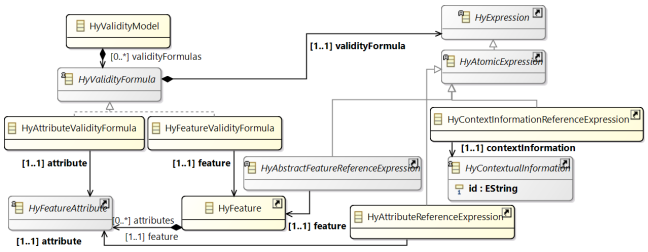

Figure 4: Meta model for Validity Formulas

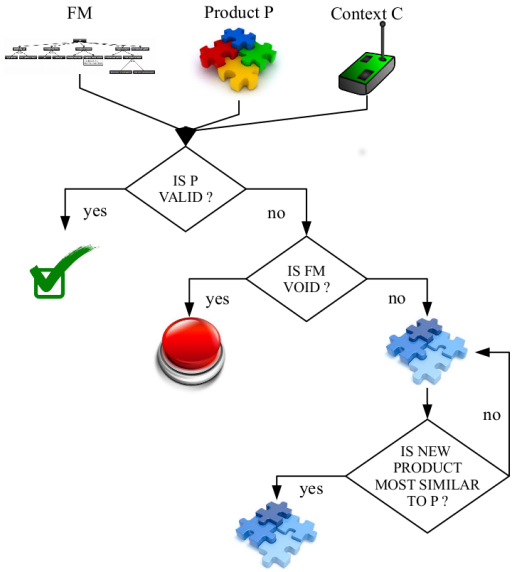

Figure 5: Work-flow of the contextual reconfigurator

To model the VFs and correlate them with features and feature attributes, we created the `HyValidityModel` which can be seen in Figure 4. With this model, we can create two different kinds of VFs: VFs for features and for feature attributes. The respective features and attributes of the feature model are referenced.

To express a concrete VF, a `HyExpression` can be utilized. With these expressions, it is possible to create arbitrary propositional formulas with the following operators: $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$. The variables of these formulas can be features or relational expressions on attributes and contextual information with the operators: $=, \neq, <, \leq, >, \geq$. Moreover, we support literal values for Integer, Boolean and enumeration types.

In addition, our meta model allows to use arithmetical expression as well. Thus, it is possible to use the following operators: +, -, *, /, %. We can use these arithmetical expressions in combination with attributes or contextual information. An example for such an expression in a VF is:

$$Road = Dry \quad \rightarrow \quad (maxSpeed * 2) \leq 600$$

All these expressions can be nested arbitrarily.

5. RECONFIGURATION ENGINE

In this section, we describe the contextual reconfigurator

HyVarRec. As depicted in Figure 5, the contextual reconfigurator requires three inputs: the FM, the current configuration P of the remote device, and the current values of the contextual information C. The primary function of the contextual reconfigurator is to check if a configuration P is valid w.r.t. the contextual information C.

Given a FM and a configuration C, let us say that the FM is void w.r.t. C if no valid configuration exists for C. If P is valid or the FM is void w.r.t. C then HyVarRec informs the user about the validity or the fact that no valid configuration exists respectively. Otherwise, in case P is not valid and the FM is not void, HyVarRec tries to produce a new valid configuration P' of FM considering the contextual information C.

Following the methodology presented in Section 4, we transformed the FM into variables and constraints, which are used as input for the reconfigurator. As we are also interested in finding a valid configuration that is most similar to the initial one, we have encoded the problem into a COP. In our setting, the reconfigurator tries to find a solution that minimizes the differences with the initial configuration. This means that the reconfigurator minimizes the number of feature removals needed to transform P into P'. In case of ties between valid configurations, the reconfigurator selects the configuration P' that has most attributes in common with P.

HyVarRec requires as input (i) a file eliciting the structure of the FM, attributes, their constraints, and a configuration and (ii) a file containing the contextual information. Thus, HyVarRec may even be used with formats other than the previously presented meta models.

HyVarRec is an anytime solver: it first produces a valid configuration and then proceeds in finding those more similar to the initial configuration. The optimization proceeds in phases: If the initial configuration is not valid, a new valid configuration is searched for. When a valid configuration is found, additional constraints are imposed to search only valid configurations that are more similar to the initial one. The process terminates when the most similar configuration is found. In this way, the reconfigurator permits users to interrupt the computation as soon as one good-enough valid configuration is found, even though it might not be the best possible configuration. This is also useful for handling large and complex SPLs when one may have limitations on the computational resources or time constraints.

HyVarRec relies on *MiniSearch* [25] to conduct the search and on *MiniZinc* [21] for the definition of the constraints. We chose to use MiniZinc since nowadays it is the most supported language to define CSPs: a large variety of solvers can process it and therefore speed up the computation of the solutions. Moreover, we chose to use MiniSearch because it is the only framework we are aware of that explicitly addresses the implementation of general search strategies on top of MiniZinc solvers.

For every feature of the FM, HyVarRec introduces a unique Boolean variable to represent if the feature was selected or not. Similarly, a unique integer variable is introduced for every attribute. The constraints related to the FM and the VFs are then translated into the corresponding constraints in MiniZinc. The translation process is straightforward since MiniZinc natively supports all the logical and mathematical operators of both the FM structural constraints and VFs.

To check if the initial model is valid, HyVarRec forces the feature and attribute variables to be equal to their values in the initial configuration. At this point, if all the constraints are satisfied, HyVarRec outputs that the configuration is valid. Otherwise, it removes the constraints that set the variables to their initial values and starts to search for a solution. If a solution is found, its equivalent configuration is printed and new additional constraints are posted to search only for solutions that represent configurations more similar to the initial one. This step is repeated until no new solution is found. When this happens, the last printed solution represents the most similar configuration to the initial one and, therefore, HyVarRec can stop the search.

HyVarRec uses the Gecode constraint solver [12] because it implements natively the incremental API needed by MiniSearch to post additional constraint without restarting the solving process from scratch. However, other MiniZinc solvers that do not support the recently defined MiniZinc incremental API could still be used at the price of restarting their engine during the optimization process. Hence, the majority of the other state of the art solvers (e.g., the lazy constraint solver Chuffed [9], the java based Choco [24], the SAT based MiniSatid [18]) can be easily integrated in HyVarRec by simply paying the price of some solver restarts.

The output of HyVarRec is a textual representation of the features and the attribute values of the valid configuration. These can be converted and processed by external visualizations tools.

6. CASE STUDY

In this section, we describe how HyVarRec has been applied on the running example introduced in Section 3. In particular, we have two relevant scenarios: the implementation of the emergency call and the maximum settable speed of the `Adaptive Cruise Control`.

As can be seen from Figure 2, the emergency call is dependent on the country in which the car is located. In Europe, it is the `eCall Europe` service, which relies on positioning data of a `GPS` system. In Russia, the `ERA/GLONASS Russia` service relies on positioning data provided by the `GLONASS` satellite system. The position of the car is represented with the contextual information `Location`, which currently supports `Russia` and `Europe`. Thus, `eCall Europe` can only be selected if the car is located in Europe and `ERA/GLONASS` can only be selected if the car is located in Russia.

For the feature `Adaptive Cruise Control`, the attribute `maxSpeed` represents the maximum speed settable by the driver. The maximum settable speed depends on two factors: the position of the car (e.g., in Russia, the speed limit is $110km/h$) and the state of the road, represented by contextual information `Road`. To reduce the accident risk, the maximum speed of the `Adaptive Cruise Control` is limited if the `Road` is `Wet` or `Icy`. Thus, if the `Road` is `Wet`, the `Adaptive Cruise Control` can only be selected, if `maxSpeed` is less than or equal to 160 km/h. If the `Road` is `Icy`, `maxSpeed` has to be less than or equal to 100 km/h.

A brand new car is pre-set with the following initial configuration on release: `eCall Europe`, `GPS`, `Adaptive Cruise Control` with `maxSpeed = 200`, `Fast Front Distance Sensor`, and `Infotainment System`.

Let us consider the case in which the car is in Europe and the used road is dry.

To check if the configuration is compatible with the current context, we can run HyVarRec. In this case, it accepts

the configuration without proposing any changes as all the VFs and the FM constraints are satisfied.

Now, suppose that it starts to rain making the road wet (i.e., the context `Road` changes from *Dry* to *Wet*). This modification makes the initial configuration invalid as the maximum speed of the car is too high for such road conditions. In this case, the reconfiguration engine detects the invalidity of the configuration and suggests, as a minimal modification, to lower the `maxSpeed` attribute to 160 without changing any additional feature.

Let us consider the case that the driver takes the car to Russia for a holiday. The change of location invalidates the configuration as it makes the VF associated to `eCall Europe` unsatisfiable. To obtain a valid configuration in this scenario, HyVarRec selects the feature `ERA/GLONASS Russia` as `Emergency Call` is a mandatory feature and the VF $Location = Russia$ evaluates to true by the new context. The effect of selecting `ERA/GLONASS Russia` causes also the deselection of `GPS` and triggers the selection of the navigation system `GLONASS`. Additionally, to satisfy the VF $(maxSpeed \leq 160) \wedge (maxSpeed \leq 110)$, the reconfiguration engine will set the `maxSpeed` to 110. Once all changes are computed, the reconfiguration engine can trigger the update of the car, bringing its configuration into a valid state.

As a final note, we would like to underline that the introduction of VF can cause the FM to not allow valid configuration for some context. For example, considering our running example, this may happen if the car goes into the USA. Indeed, this context does not admit any valid configuration as the mandatory `Emergency Call` feature can not be selected ($Location \neq Russia$ and $Location \neq Europe$). In such a situation, the output of the reconfiguration engine may be used to initiate proper actions such as informing the driver about the missing emergency call option in the USA or switch to a default configuration and send a report to the manufacturer to inform about an incomplete FM.

To prove the feasibility of our approach, all these scenarios have been modeled and successfully verified with HyVarRec. Due to the limited size of the considered FM, the computation of the reconfiguration was instantaneous.

We would like to underline that the problem solved by HyVarRec is from the complexity point of view an NP-hard problem as even only checking if a FM is void is NP-hard. Hence, from the theoretical point of view, HyVarRec can require a huge amount of time to solve some instances and this is unavoidable. However, as witnessed for example in the Linux community where NP-hard problems are concretely solved to compute the packages to install [1], we believe that HyVarRec can be used in practice for the majority of real-world problems. Nevertheless, as a future work, we plan to conduct more tests with bigger FM to check the scalability of the entire approach.

7. RELATED WORK

A wide range of approaches for the development of context-aware software exists in the literature. Among the works using SPL, one of the closest to our approach is [14]. Features are constrained to contextual information via the Context Variability Model. The combination of the Context Variability Model and the FM results in what the authors called a Multiple Product Line FM. The contextual information is not captured as in our case within the original FM but imposed as additional cross-tree constraints. Similarly, in [2, 26], both the context and the variability model are captured by using two distinct feature models that are connected using rules that establish how to configure a system based on contextual information. These FMs could be composed using the approach presented in [3]. In [11], FMs are enriched with contextual information but, differently from our case, their connection with the features is given using some external rules or constraints. In [20], the contextual information used to model adaptation of applications is described by an ontology representing a global context model. Moreover, local context models tailored to the specific needs of a particular application are defined by the authors as a view over the global context in the form of a feature model. Rules are then used to generate the feature model from the global context.

Our approach deviates from these ones as we explicitly connect the contextual information with the features. In this way, the developer can better visualize, maintain, and adapt the SPL. We argue that starting from a complex contextual model, such as the ones in [2, 11, 14], it is always possible to encode the relevant information of its concrete instances into our representation. Therefore, our approach is not incompatible with complex contextual models that can indeed still be used, provided that, in a pre-compilation phase, their concrete instances are encoded into a map of identifiers-values.

In [23], a model-driven engineering approach for transforming a generic feature model according to a context model was proposed. This is a completely different approach from ours. We do not handle and manipulate FM but instead we introduce into existing FM relations with the contextual information. Thus, with our methodology, the original FM will remain unmodified.

8. DISCUSSION

As previously mentioned, the introduction of the VF can cause the FM to be void for some context. Usually, FM are designed to not be void and this check can be performed at compile time by solving a CSP. This can naturally be lifted up to the context-aware FM by ensuring that, for every context, the FM admits a valid configuration. This validity extension may, however, be too demanding. For instance, consider the case of a context that can never happen (e.g., an `Icy` road on a country where the temperature never goes below 0). In this particular case, if the context is unrealistic, it does not matter if the FM admits a valid configuration or not. Therefore, when context is added into the FM, it may make more sense to check if the FM admits a valid configuration for every realistic context. For this reason, following the approach in [27], we are considering the definition of a development framework where the developer is asked iteratively to specify what the realistic/unrealistic contexts are. The check of the validity of the FMs for the realistic context is done in parallel to warn the developer in case of inconsistencies (i.e., when a realistic context does not admit a valid configuration). However, from a computational point of view, this is more complex than checking the validity of a single FM because the introduction of the quantifiers for the realistic contexts makes the problem Σ_2^P-complete (which unless P = NP, is more complex than the FMs validity check which is NP-complete). Therefore, to concretely solve this problem, following [27], we are considering the use of SAT Modulo Theory (SMT) solvers as they

natively support quantification or, if more expressive power is needed, we plan to extend the more general CP solvers with quantifiers. This techniques can also be used to prove other interesting properties. For instance, with a different search strategy, it may be possible to find dead features (i.e., when a feature can not be selected) or false variables (i.e., when a feature must always be selected).

Another important issue for the development of a general framework for reconfiguration similar to the one presented in Section 3 is the performance of the reconfiguration task. There are different directions for improving the reconfiguration engine, e.g., the use of portfolio techniques [4] or the use of local search algorithms [15]. However, as the problem is NP-hard, some backup solution may also be required if the reconfiguration takes more than the allowed time. In this case, e.g., a default configuration may be used and a warning may be sent to the developer that could investigate further why the reconfiguration took so much time.

When dealing with reconfiguration, we may face the problem that there are too many configurations we can choose from. To restrict the possibilities, HyVarRec tries to choose the configuration that is most similar to the initial one. In general, it may be even more interesting to allow users to choose their preferred configuration by a specific metric. The introduction of the context and the dynamicity of the reconfiguration add an extra dimension to the already difficult task of gathering user preferences. Thus, this poses additional challenges for approaches such as [19] for the visualization and elicitation of the optimal criteria for selecting a configuration. Indeed, the user preferences now may be related also to contextual information and this requires new kinds of visualization tools to express the preferences when the context is varying.

9. CONCLUSION

In this paper, we introduced a methodology to dynamically reconfigure an SPL based on contextual information. To this end, we created a concept to represent contextual information which is limited to the essential information. We introduced the notion of Validity Formula (VF): propositional formulas on contextual information, features, and feature attributes to constrain the feature selection. To provide the relevant information of the VFs at the relevant location to the SPL developers, we correlated the VFs with their respective features.

Additionally, we define HyVarRec, a reconfigurator tool that, starting from the FM, the current configuration and the current contextual information, checks if it is necessary to create a new configuration and if so attempts to propose one that is most similar to the original configuration given in input. We demonstrated the feasibility of our methodology using a case study based on an excerpt of the SPL of our industrial partner.

For future work, we are interested in the extension of HyVarRec to detect, at compile time, when and for which context the FM admits no solution or other FM related properties such as dead features or false variables. Moreover, we are interested in allowing HyVarRec to support user defined preferences. Finally, we are interested in the integration of HyVarRec into a more general and inclusive tool for software reconfiguration that, starting from concrete devices and FM, is able to configure and update their software.

Acknowledgments

This work was supported by the European Commission within the project HyVar (grant agreement H2020-644298).

10. REFERENCES

[1] P. Abate, R. D. Cosmo, R. Treinen, and S. Zacchiroli. MPM: a modular package manager. In *CBSE*, pages 179–188. ACM, 2011.

[2] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J.-P. Rigault. Modeling Context and Dynamic Adaptations with Feature Models. In *4th International Workshop Models@run.time*, page 10, 2009.

[3] M. Acher, B. Combemale, P. Collet, O. Barais, P. Lahire, and R. France. Composing your compositions of variability models. In A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. Clarke, editors, *Model-Driven Engineering Languages and Systems*, volume 8107 of *Lecture Notes in Computer Science*, pages 352–369. Springer Berlin Heidelberg, 2013.

[4] R. Amadini, M. Gabbrielli, and J. Mauro. A multicore tool for constraint solving. In *IJCAI*, pages 232–238. AAAI Press, 2015.

[5] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A Survey. *Comput. Netw.*, 54(15):2787–2805, 2010.

[6] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *IJAHUC*, 2(4):263–277, 2007.

[7] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010.

[8] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In O. Pastor and J. Falcão e Cunha, editors, *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 491–503. Springer Berlin Heidelberg, 2005.

[9] Chuffed. https://github.com/geoffchu/chuffed.

[10] R. de Lemos et al. Software Engineering for Self-Adaptive Systems: A second Research Roadmap. volume 10431 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010.

[11] P. Fernandes, C. Werner, and E. Teixeira. An Approach for Feature Modeling of Context-Aware Software Product Line. *J. UCS*, 17(5):807–829, 2011.

[12] GECODE. http://www.gecode.org/, 2015.

[13] H. Gomaa and M. Hussein. Dynamic Software Reconfiguration in Software Product Families. In *PFE*, volume 3014 of *LNCS*, pages 435–444. Springer, 2003.

[14] H. Hartmann and T. Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *SPLC*, pages 12–21. IEEE Computer Society, 2008.

[15] J. Hromkovič. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[16] K. Kang. *Feature-oriented Domain Analysis (FODA):*

Feasibility Study ; Technical Report CMU/SEI-90-TR-21 - ESD-90-TR-222. Software Engineering Inst., Carnegie Mellon Univ., 1990.

[17] A. K. Mackworth. Consistency in Networks of Relations. *Artif. Intell.*, 8(1):99–118, 1977.

[18] Minisatid. https://github.com/broesdecat/Minisatid.

[19] A. Murashkin, M. Antkiewicz, D. Rayside, and K. Czarnecki. Visualization and exploration of optimal variants in product line engineering. In *SPLC*, pages 111–115. ACM, 2013.

[20] S. Neskovic and R. Matic. Context modeling based on feature models expressed as views on ontologies via mappings. *Comput. Sci. Inf. Syst.*, 12(3):961–977, 2015.

[21] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a Standard CP Modelling Language. In *CP*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

[22] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[23] T. Possompès, C. Dony, M. Huchard, and C. Tibermacine. Model-Driven Generation of Context-Specific Feature Models. In *SEKE*, pages 250–255. Knowledge Systems Institute Graduate School, 2013.

[24] C. Prud'homme, J.-G. Fages, and X. Lorca. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2014.

[25] A. Rendl, T. Guns, P. J. Stuckey, and G. Tack. MiniSearch: A Solver-Independent Meta-Search Language for MiniZinc. In *CP*, volume 9255 of *LNCS*, pages 376–392. Springer, 2015.

[26] N. Ubayashi and S. Nakajima. Context-aware feature-oriented modeling with an aspect extension of vdm. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 1269–1274, New York, NY, USA, 2007. ACM.

[27] Y. Xiong, H. Zhang, A. Hubaux, S. She, J. Wang, and K. Czarnecki. Range Fixes: Interactive Error Resolution for Software Configuration. *IEEE Trans. Software Eng.*, 41(6):603–619, 2015.