Reducing feature models to improve runtime adaptivity on resource limited devices

Conference Paper · September 2012

DOI: 10.1145/2364412.2364435

5 authors, including:

Karsten Saller
Technische Universität Darmstadt
11 PUBLICATIONS **34** CITATIONS
SEE PROFILE

Sebastian Oster
Method Park
24 PUBLICATIONS **364** CITATIONS
SEE PROFILE

Julia Schroeter
Technische Universität Dresden
11 PUBLICATIONS **103** CITATIONS
SEE PROFILE

Malte Lochau
Technische Universität Braunschweig
51 PUBLICATIONS **439** CITATIONS
SEE PROFILE

Reducing Feature Models to Improve Runtime Adaptivity on Resource Limited Devices

Karsten Saller*
Real Time Systems Lab
TU Darmstadt
saller@es.tu-darmstadt.de

Sebastian Oster
LOGICA
sebastian.oster@logica.com

Andy Schürr
Real Time Systems Lab
TU Darmstadt
schuerr@es.tu-darmstadt.de

Julia Schroeter†
Institute for Software- and Multimedia-Technology
TU Dresden
julia.schroeter@tu-dresden.de

Malte Lochau
Institute for Programming and Reactive Systems
TU Braunschweig
lochau@ips.cs.tu-bs.de

ABSTRACT

Mobile devices like smartphones are getting increasingly important in our daily lifes. They are used in various environments and have to dynamically adapt themselves accordingly in order to provide an optimal runtime behavior. Naturally, adapting to continuously changing environmental conditions is a challenging task because mobile devices are always limited in their resources and have to adapt in real-time. In this paper, we introduce an approach that enables resource limited devices to adapt to changing conditions using dynamic software product lines techniques. Therefore, feature models are reduced to a specific hardware context before installing the adaptive mobile application on the device. This reduces the amount of possible configurations that are compatible with the device and, thereby, minimizes the costs and the duration of an adaptation during runtime.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.2.9 [**Software Engineering**]: Management—*Adaptive Software Systems*; D.2.9 [**Software Engineering**]: Management—*Software Configuration Management*

*Author supported by the German Research Foundation, Research Group 733, "QuaP2P: Quality Improvement of Peer-to-peer Systems".

†Author supported by the European Social Fund, Federal State of Saxony and SAP AG in the doctoral project #080949335.

Keywords

Dynamic Software Product Lines, Feature Models, Adaptive Systems, Context-Awareness

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

Emerging domains such as ubiquitous and pervasive software systems are becoming more and more widespread. Modern computing and network environments demand a high degree of adaptability [10]; sensor nodes, e.g., have to adapt their transmission granularity to the remaining battery power to enhance their life-time. These systems must dynamically adapt to continuously changing environmental conditions during runtime. Such Dynamic Adaptive Systems (DAS) are complex software systems with a high degree of variability in both requirements and resource constraints [2]. They are able to adapt their properties and configuration at runtime in response to dynamically varying needs and constraints.

Covering a multitude of different devices and platforms and, therefore, facing both diverging properties and resource requirements, the variability has to be handled at multiple points during the lifetime of a DAS. By specifying a DAS as a software product line (SPL) the heterogeneity between devices or platforms is managed by deploying specific product instances of a DAS on each device. Feature models [12] are used to describe the variability in an SPL, specifically how features of an application are related to each other and which constraints are specified for them. Each feature represents either a logical group of requirements or a system property that is relevant to some stakeholder [3, 9]. A configuration consists of features that were selected according to the variability constraints defined by the feature model [7].

Standard SPL engineering usually deals with static environments, where a system is configured at design time to the needs of a customer. In such a scenario, an adaptation of a product configuration at runtime is out of scope. Dynamic SPLs (DSPLs) have been proposed to handle such runtime adaptivity with SPL engineering methodologies [10]. DSPLs

are characterized by their abilities to handle configuration and binding of features at runtime as well as to deal with changes in the functional or qualitative requirements [10]. Using feature models to describe the variability, it is possible to adapt to the environmental conditions (context) by continuously reconfiguring the DAS to the requirements a context constitutes [6]. At runtime, a feature model is used to compute a configuration to derive a plan how the components of a DAS have to be bound and reconfigured.

However, applying DSPL based methodologies to resource limited devices is a difficult task. Specifying constraints using a feature model requires a constraint solver to calculate a configuration that matches both the specified constraints and the requirements. The more features and constraints a feature model contains, the more complex becomes the process of solving a configuration. This is even intensified if we focus on mobile and adaptive systems: any change in the context may trigger a (re-)configuration. Thus, high mobility will drain available resources, like battery power.

In this paper, we introduce our feature model reduction approach to improve the runtime behavior of a DSPL oriented configuration process. We focus on resource limited, networked devices in distributed environments, e.g., smartphones or sensor nodes. Identifying several major challenges for the development process in DAS, we propose an approach that deals with the heterogeneity and resource limitations of mobile devices. Providing a partial configuration that describes a specific hardware context of a device, we are able to remove all features that are not compatible with this configuration. By extracting the features that are variable, i.e., that may be part of a configuration after this pre-configuration or not, we reduce the space of possible configurations for a device. This improves the configuration process at runtime in both responsiveness and resource consumption.

2. RELATED WORK

Czarnecki et al. were the first to propose the approach of a staged configuration using feature models [8]. The authors argue that the derivation of a product configuration using feature models may be done in stages, where each stage eliminates possible configuration choices. With a given input model, each stage yields a subset of the input model thereby forming a specialized output feature model. The process of a staged configuration describes successive specialization steps followed by configuration steps using the most specialized feature model. We leverage the process of a staged configuration to minimize the resource consumption when reconfiguring mobile devices at runtime. This approach can be described as applying a partial configuration until there is no more variability left. In contrast to the proposed staged configuration, our approach neither specializes nor generalizes the constraints of a feature model. Instead our reduced feature model is always consistent with the specified constraints in the original feature model.

Bencomo et al. investigated component-based technologies for runtime adaptivity using a component model called OpenCOM [2]. OpenCOM offers component frameworks to the developers, for both application and middleware platform. The adaptive behavior is specified by reconfiguration policies in the form of event-do-action rules. In this case, an action describes architectural changes in the component model, i.e., it binds components to other components during runtime. Every possible variation of active components and their relations is called a configuration. Using context-aware monitors, relevant information is captured from the environment which triggers the event-do-action properties. The authors applied their approach to wireless sensor nodes within a case study for a flood warning system. Possibilities for an optimization, like a pre-configuration, are proposed but not yet realized.

Another component-based approach that deals with adaptivity in ad-hoc systems is proposed by Pepper et al. [15]. The authors specifically address the problem of resource limitation of mobile devices in an ad-hoc network and provide an approach to handle the quality of services. Services are composed by linking service units which are characterized by their functionality and quality. The attributes are used to describe the non-functional qualitative nature of a feature. Including quality and functional properties, the resulting matching problem to find an optimal configuration suitable for the stated requirements (best service vs. few resources) is considered to be a major challenge. Pepper et al. proposed several challenges and possibilities how to deal with qualitative characteristics and how the solving may be improved in a fair manner but no solution is provided yet.

There are several approaches which integrate contextual aspects in variability modeling. Ali et al. studied how context can influence variability [1]. Similar to our approach they identified different variation points in which the context has to be considered, e.g., at design time and at runtime. Similarly, Hartmann et al. proposed the concept of a context variability model [11]. Based on the context they derive multiple context specific product lines. Both approaches combine goal models with feature models to reason about the requirements in a specific context. In our approach, we propose to use separate feature models and to define a functional mapping, i.e., require or exclude relations, between the features in both feature models. Thus, we separate the modeling of the variability and contextual requirements.

3. CASE STUDY

This section introduces a case study which we use to exemplify our approach in the following sections. The feature model describes a small subset of a software product line for DAS, focusing on mobile devices.

The spectrum of devices in a DAS is characterized by their heterogeneity, e.g., the supported connection types, provided hardware sensors, and available memory. Figure 1 shows an example of a Software Product Line (SPL) for smartphones, specified in the FODA notation for feature modelling [7, 12].

`Connection` describes how the devices are connected to an arbitrary network. This feature is a *mandatory feature*, which means that it is contained in every possible product configuration. A connection may either be `Infrastructure` based (a centralized one-hop connection), like a connection via `GSM` or via an `Wi-Fi AP` (access point). Furthermore, the connection may be a wireless `Ad-Hoc` based connection [4] and, thus, providing only access via decentralized multi-hop networks. In ad-hoc environments, every device is both a sender and a receiver, and operates as a routing point for all devices in its range. As a consequence, it is not possible to maintain an infrastructure and wireless ad-hoc based connection at the same time. To specify such an exclusive-or relation, *alternative-groups* are used. Such a group specifies that exactly one feature in this group has to be selected

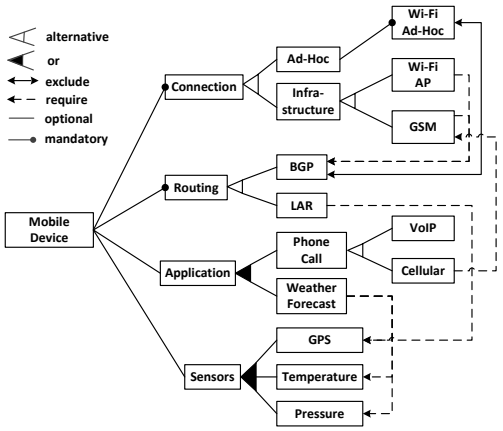


Figure 1: Feature Model for Mobile Devices

when the parent feature is part of the configuration.

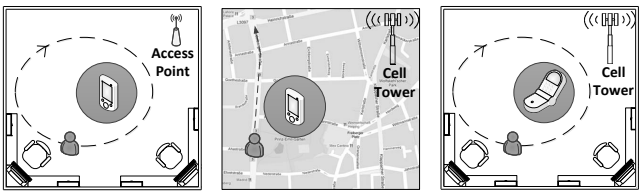
Depending on the environment, different `Routing` protocols have to be used to provide an efficient connection. In our case study, we have an alternative-group of two different routing protocols: (i) `BGP` (Border Gateway Protocol[1]) is an Internet Protocol (IP) based routing protocol which is commonly used in infrastructure based networks like the Internet. Hence, it is not compatible with a wireless ad-hoc connection. Such constraints are modeled via different types of *cross-tree constraints*. In this case of incompatibility between two features an *exclude* relation is defined. (ii) `LAR` (Location Aided Routing [14]) is a routing protocol for wireless ad-hoc networks that uses the geographic location of the devices, provided by a `GPS` sensor. Instead of flooding the message to all devices in the neighborhood, the message can be directed in a geographic direction. `LAR` cannot operate without a `GPS` sensor. Such a dependency is modeled via the cross-tree constraint relation *require*.

To provide flexibility in the targeted hardware platforms, sensors are considered to be *optional features* and, therefore, need not be part of every possible configuration. However, if sensors are available on a specific device, we support multiple types of sensors, e.g., sensors for `GPS`, `temperature`, and `pressure` monitoring. Once the parent feature `Sensor` is part of the configuration, multiple but at least one of the sub-features have to be contained in the final configuration. Such a group of features is called an *or-group*.

Two optional example `Applications` are also contained in an or-group. The feature `VoIP` (Voice over IP) describes the ability to make a `Phone Call` over the Internet. To provide its functionality, `VoIP` requires `BGP` to be part of the configuration. In addition to that, it is always possible to make a `Cellular` based phone call as long as the `GSM` feature is part of the product configuration. The application feature `Weather Forecast` requires all sensors, i.e., `GPS`, `Temperature`, and `Pressure`.

For our case study we introduce two different kinds of mobile phones: a *smartphone* of the most recent generation and a classic *mobile phone*, in two different environmental situations: Both phones are mobile and can move from one context environment to another, but differ in their capabilities. The smartphone is capable of networking via GSM and wi-fi; the mobile phone has no means of networking but is capable to communicate via GSM. For example, Figure 2 shows three different contextual scenarios: moving from (a) an office room to (b) walking the streets in a city. The third scenario (c) is comparable to the first scenario (a), but a mobile phone is used instead of a smartphone.

[1]https://bordergatewayprotocol.net/



(a) Office Ad-Hoc (b) City GSM (c) Office GSM

Figure 2: Sample Contextual Situations

4. CONTEXT AWARENESS WITH DSPL

In this section we point out the importance of the different contextual categories in the lifetime of a DAS. Afterwards we point out major challenges in the development of context-adaptive applications for DAS, derived by the case study and the different contextual categories.

4.1 Contextual Categories

To bring the concept of context-awareness on mobile DAS is a major issue in recent research [17]. One solution is to equip a DAS with a so-called context model that describes the characteristics of possible runtime environments.

The context specifies certain requirements for the functional and qualitative aspects of a software product before and at runtime [1, 11]. A change in the context triggers a change in the configuration of the product to adapt it accordingly [6]. A functional requirement can be mapped to a feature or a set of features. Qualitative characteristics have to be assigned to every feature. Features, which provide the same functionality, can then be compared based on their qualitative characteristics to meet the qualitative requirements. For example, considering situation (a) in our case study, it is possible to make either a VoIP based or cellular based phone. But, if we want to use the cheapest alternative, we will choose VoIP because it is cheaper than a cellular based phone call.

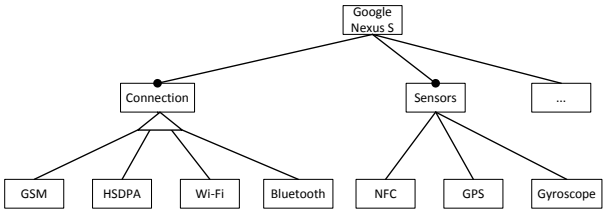


Figure 3: Hardware Features for a Google Nexus S

We identified several contextual categories that may be used for a context aware minimization of the feature model and the reconfiguration at runtime: The *hardware context* is directly related to each device, describing provided hardware characteristics, e.g., installed sensors or CPU. There-

fore, this type of context is static for each device. The hardware context provides both information regarding incompatibilities to and requirements for specific features which are part of an SPL. Such contextual information can be specified with a feature model, e.g., the hardware context of the provided hardware features of a smartphone is depicted in Figure 3. The depicted hardware context describes an extract of the smartphone product line for a Google Nexus S[2]. This feature model defines a pre-selection of features that are provided by such a kind of device. Using this knowledge allows to derive a device specific configuration of a DAS. Thus, every feature that is either required or incompatible with the installed hardware has to be statically bound into or completely removed from every configuration of a software product for such a device.

Other contextual categories that may trigger a reconfiguration are based on the *location*, e.g., in an office or on a street, the *time*, e.g., silent mode at night, the *available resources*, e.g., the remaining battery power, or the used *interfaces*, e.g., connected to a docking station. This shows, that DAS are in a continuous struggle to adapt the functionality of a device to available services and resources.

4.2 Challenges

The presented case study comprises several challenges that have to be addressed when developing applications for mobile devices which are capable to adapt themself to their environment.

1. *Heterogeneity.* DAS include a multitude of devices, basically any device that is capable to adapt itself to its environment. This includes for example sensor nodes [2], autonomic pervasive systems [5], or multimedia capable devices [15] like smartphones and tablets. These devices differ in their provided features, e.g., in their sensing capabilities, operating systems, or equipped hardware. The composition of different features is specific to every type of device. Hence, the capabilities and requirements of each device are different depending on the provided features.

2. *Resource limitation.* Hardware constraints are a common issue for developing applications for mobile devices [15]. Every device is limited by the installed hardware, e.g., the CPU, memory, and battery. In our case study, we have two rather oppositional devices: a smartphone and a more resource limited mobile phone. The computation of a configuration process consumes resources. To improve the resource consumption, the amount of computations has to be minimized and the process of computing a configuration has to be optimized, e.g., by reducing the computational complexity.

3. *Quality of Service.* Deriving a configuration using a feature model is usually limited to the functional aspects of the features. To support requirements regarding the quality of a provided functionality, it is necessary to include features that provide the same functionality, and are thereby exchangeable, but show different runtime behavior. Benchmarking allows us to categorize features in their non-functional runtime behavior for the considered quality aspects [15]. To annotate qualitative characteristics, feature models have to be extended using attributes [13] such that requirements like "the responsiveness has to be below four seconds" can be specified. However, a drawback of this approach is that the process of solving the constraints and deriving a configuration gets more complex and, thus, consumes more resources.

4. *Derive context.* To provide context awareness, the raw information which characterize the state of the environment have to be collected first [17]. This is usually done by using sensors and monitoring mechanisms. After a snapshot of the environment is taken, the collected information has to be analyzed to derive the needed information to trigger a local reconfiguration. For such a reconfiguration, the current state of the device configuration, a global goal, and the new environmental conditions and requirements have to be known [1].

5. *Runtime adaptivity.* Looking at the scenarios in our case study, an application has to compensate a change of context when moving from one situation to another (c.f. Section 3: moving from (a) to (b)). To avoid any loss of data and to ensure responsiveness of the device, the process of adaptation has to be seamless and continuous. Using a DSPL based engineering approach mechanisms can be provided that support the adaptivity and autonomy of an application [6]. Considering the previous challenge 2, the process of adaptation has also to be efficient and to consume a minimal amount of resources.

In the following sections, we will provide an approach that addresses the first and the second challenges. The challenges of handling non-functional characteristics, context derivation, and the process of runtime adaptation are considered to be future work in our research.

5. IMPROVING RUNTIME ADAPTIVITY BY FEATURE REDUCTION

In this section we introduce our approach to improve the computation of a configuration at runtime. Therefore, a feature model for a large domain is reduced until there is only a minimal set of variable (unbound) features left. By applying a partial configuration, this feature model is intended to constitute a pre-configuration. Such a partial configuration is given by the contextual requirements and may be used to restrict the variability in a feature model. Figure 4 depicts this process: starting with a large domain feature model, a partial configuration is selected before a restricted, reduced feature model is deployed on a phone. The reduced feature model contains the needed variability to compute a configuration according to the dynamically changing requirements, continuously during runtime.

5.1 Reducing a Feature Model

A feature model can be depicted as a tree of features, c.f. Figure 1, containing features as nodes, and constraints as parent-child relations and cross-tree constraints. Such a feature model is also specifiable as a set of propositional logic formulas, using $\vee$ (disjunction), $\wedge$ (conjunction), and $\neg$ (negation). Thus, a *feature model* $FM = (\mathcal{F}, \mathcal{C})$ is defined as a finite set of features $\mathcal{F} = \{f_1, \ldots, f_n\}$ and $\mathcal{C}$ as a single propositional (three valued) logic formula over $\mathcal{F}$.

[2]http://www.google.com/nexus/#/tech-specs

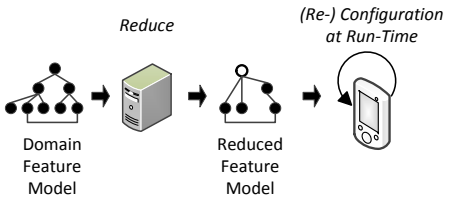


Figure 4: Minimization Process for DSPLs

A feature model represents a set of *configurations* Γ. Each *configuration* $\gamma \in \Gamma$ is defined as an assignment of the three-valued logic truth values in $\mathcal{F}$:

$$\gamma \in \Gamma : \mathcal{F} \rightarrow \{0, ?, 1\} \tag{1}$$

- 0 = *false* (feature is bound to value 0 , i.e., deselected or dead)
- $?$ = *unknown* (feature is unbound, i.e., neither bound to 1 or 0)
- 1 = *true* (feature is bound to value 1, i.e., selected or alive)

to the features in $\mathcal{F}$.

Bound-alive features are characterized by their necessity on a device or in a certain context. These and all implicitly required features are always part of a product configuration for a specific context and may not be removed. For example, the feature `Connection` in Figure 1 is considered to be a bound-alive feature for the hardware context in Figure 3 because the targeted device is equipped with several types of connections.

In contrast to a bound-alive feature, *bound-dead features* are excluded from a configuration, e.g., if they are not compatible with a certain context. Therefore, they are removed from the software product line that is specific to a certain context. For example, the feature `Temperature` in the feature model in Figure 1 is removed from the feature model as soon as it is deployed on a smartphone that is characterized by the hardware context depicted in Figure 3.

Unbound features may not be required by bound-alive features or require bound-dead features. In a further reduction of a configuration, they can still be selected as a part of the configuration.

A configuration $\gamma \in \Gamma$ thus defines a substitution on a formula $\mathcal{C}$ with

$$\mathcal{C}' := \mathcal{C}[\gamma]$$

where $\mathcal{C}'$ is derived from $\mathcal{C}$ by replacing all occurrences of a feature f in $\mathcal{F}$ by

- 1 iff $\gamma(f) = 1$
- f iff $\gamma(f) = \ ?$
- 0 iff $\gamma(f) = 0$

A configuration gamma is called *complete* iff

$$\nexists f \in \mathcal{F} : \gamma(f) = \ ?$$

Analogously, a configuration is called *partially complete* iff

$$\exists f \in \mathcal{F} : \gamma(f) = \ ?$$

Thus, a partial configuration is similar to a specialization of a configuration. Although the variability is reduced in this process, a partial configuration is still containing features that are unbound and, therefore, are variable.

The propositional formulas can be constructed as a conjunction of implications from optional or mandatory child-parent relations, implications from parents to groups and additional constraints as require or exclude constraints. A configuration is invalid if the feature assignment is not compatible with the constraints $\mathcal{C}$ defined in FM.

The *set of all (partially) consistent configurations* of $FM = (\mathcal{F}, \mathcal{C})$, i.e., the set of all substitutions that do not (yet) map the formula $\mathcal{C}$ onto an antinomy is given by

$$\Gamma_{\mathcal{C}} := \{\gamma \in \Gamma \mid \mathcal{C}[\gamma] \nvdash 0\}$$

Let $FM = (\mathcal{F}, \mathcal{C})$ and $FM' = (\mathcal{F}', \mathcal{C}')$ be two feature models and $\gamma \in \Gamma$ be a configuration of FM. A feature model FM' is a *reduction* of FM w.r.t. to the configuration γ iff

$$\begin{aligned} \mathcal{F}' = \{\ f \in \mathcal{F} \mid \gamma(f) = \ ? \\ \wedge\ \mathcal{C}' \vdash \mathcal{C}[\gamma] \\ \wedge\ \mathcal{C}[\gamma] \vdash \mathcal{C}'\ \} \end{aligned} \tag{2}$$

The reduced and, thus, smaller feature model contains only the still unbound feature set F' of the original feature model and a new usually simpler constraint $\mathcal{C}'$ which is equivalent to the original constraint $\mathcal{C}$ restricted to the remaining set of unbound features. All bound-alive features are always and all bound-dead features are never part of a configuration by definition. Thus, if an unbound feature becomes bound and part of the configuration in a further reduction or runtime-configuration, the required bound features are already part of the configuration.

THEOREM 1. *Any consistent configuration $\gamma' \in \Gamma_{\mathcal{C}'}$ of a reduced feature model $FM' \subseteq FM$ w.r.t. γ can be extended to a consistent configuration $\gamma^+ \in \Gamma_{\mathcal{C}}$ of the original feature model FM such that*

$$\begin{aligned} \forall\ f' \in \mathcal{F}' :\ \gamma^+(f') = \gamma'(f') \\ \forall\ f \in \mathcal{F} \backslash \mathcal{F}' :\ \gamma^+(f) = \gamma(f) \end{aligned} \tag{3}$$

Furthermore, any consistent configuration $\gamma^+ \in \Gamma_{\mathcal{C}}$ with

$$\forall\ f \in \mathcal{F} \backslash \mathcal{F}'\ :\ \gamma^+(f) = \gamma(f) \tag{4}$$

can be reduced to a consistent configuration $\gamma' \in \Gamma_{\mathcal{C}'}$ such that

$$\forall\ f \in \mathcal{F}'\ :\ \gamma'(f') = \gamma^+(f') \tag{5}$$

PROOF. That Theorem 1 holds can be shown by contradiction as follows:

1. Obviously, there exists at most one $\gamma^+ \in \Gamma_{\mathcal{C}}$ according to the extension of a γ' as defined in (4) ($\forall\ f \in \mathcal{F}$)
2. Let us assume that $\gamma^+ \notin \Gamma_{\mathcal{C}}$

   $\Rightarrow\ \mathcal{C}[\gamma^+] \vdash 0$

   $\Rightarrow\ \mathcal{C}[\gamma][\gamma'] \vdash 0$

   $\Rightarrow\ \mathcal{C}''[\gamma'] \vdash 0$ with $\mathcal{C}'' = \mathcal{C}[\gamma]$

   $\Rightarrow\ \mathcal{C}'[\gamma'] \vdash \mathcal{C}''[\gamma'] \vdash 0$ with $\mathcal{C}' \vdash \mathcal{C}''$ as required in (2)

   $\Rightarrow$ contradiction to $\gamma' \in \Gamma_{\mathcal{C}'}$ $\lightning$

Analogously, we can show that any consistent configuration $\gamma^+ \in \Gamma_{\mathcal{C}}$ with $\gamma^+(f) = \gamma(f)\ \forall f \in \mathcal{F}\backslash\mathcal{F}'$ can be reduced to a consistent configuration $\gamma' \in \Gamma_{\mathcal{C}'}$. □

5.2 Reducing for a Hardware Context

An application scenario for minimizing a feature model to improve the runtime adaptivity is depicted in Figure 5. In this scenario, a deploy server is equipped with a domain feature model valid for a DAS and supports multiple heterogeneous devices. Additionally, configurations which describe the hardware context of all target devices reside on this server. Therefore, the deploy server is capable to reduce the domain feature model with respect to every partial (device-)configuration on the server.

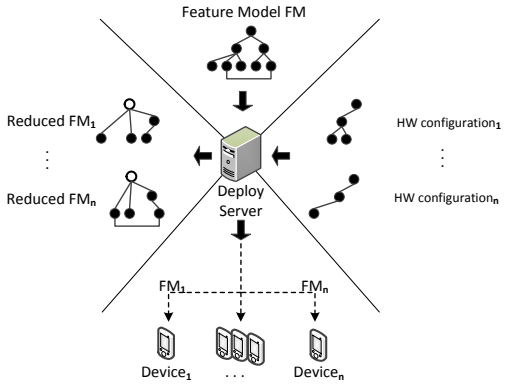


Figure 5: Deployment of reduced Feature Models

Algorithm 1 Deployment of a reduced FM'

Require: A feature model FM and a consistent partial device configuration γ_d
Ensure: γ_d specific FM' reduced w.r.t (3), (4), and (5)

```
FM' = (F', C') ← (F, C)
for all f ∈ F' do
    if (γ_d(f) ≠ ?) then
        F' ← F' \ f
        C' ← adapt(f, γ_d(f), C')
    else if (bound(f, 1, γ_d, C') then
        F' ← F' \ f
        C' ← adapt(f, 1, C')
    else if (bound(f, 0, C) then
        F' ← F' \ f
        C' ← adapt(f, 0, C')
    end if
end for
Deploy FM' to devices d | γ_d
```

5.2.1 Process of Reduction

The detailed reduction process for this deployment scenario is given in Algorithm 1 that implements our formal reduction approach from the previous Section. For a given domain feature model FM, we assume to have a configuration γ_d describing the hardware context for one specific device $d \in \mathcal{D}$. Thus, there is a configuration γ_d for every device $d \in \mathcal{D}$ that fulfills the following conditions:

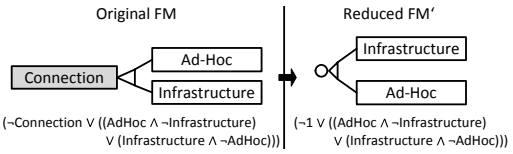
$$f \in \mathcal{F}_{HW} : \gamma_d(f) \neq\ ?$$
$$f \in \mathcal{F}\backslash\mathcal{F}_{HW} : \gamma_d(f) =\ ?$$

With a given FM and γ_d the algorithm computes a reduced FM' as follows. To derive a reduced feature model FM' for a configuration γ_d, the algorithm checks every feature f $\in \mathcal{F}$. To find unbound features for FM', the feature has to be evaluated if it can be set to bound-alive (1) or bound-dead (0) for γ_d according to the constraints in $\mathcal{C}$. This is done by an evaluation function `bound(f, v, γ_d, C')` that evaluates the feature according to its assignment with the given configuration and constraints. If a substitution of f by ¬v in the formula $\mathcal{C}$ leads to a contradiction, the feature f may only be substituted by v:

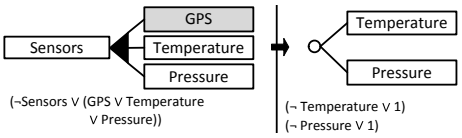
$$\text{bound(f, v, } \gamma, \mathcal{C}') :\Longleftrightarrow \mathcal{C}'[f \rightarrow \neg v] \vdash 0$$

Thus, f is implicitly bound to v and the function `bound` is evaluated to true.

Keeping the constraints consistent in both FM and in the derived FM' is a difficult task. Removing features from the original feature model implies to remove constraints related to those features. This leads to new configuration possibilities that were not intentionally specified. Thus, the constraints have to be adapted to provide a consistent reduction of FM which is done by the method `adapt(f, v, C)`. A naive implementation of `adapt` simply replaces all occurrences of f in $\mathcal{C}$ by v, i.e., `adapt(f, v, C)` $= \mathcal{C}$ [f $\rightarrow$ v]. An extensive description of all possible constraint transformations is out scope in this paper. However, we give a short description of two transformation possibilities depicted in Figure 6 for an (i) or-group and (ii) an alternative-group. If the parent feature of an alternative-group is configured to be bound-alive (grey background) it is substituted by 1 and removed from FM'. The alternative-group relation is then inherited to the root feature in FM' (c.f. Figure 6(a)). In contrast, if a feature within an or-relation is selected to be bound-alive, the group is dissolved and all other features become optional features, and, therefore, are unbound. The parent feature is selected to be bound-alive within the configuration and the optional features are linked to the root feature in FM' (c.f. Figure 6(b)).



(a) Alternative-Group, Parent Selection



(b) Or-Group, Child Selection

Figure 6: Resolving Group Constraints

After all features in FM' have been checked and configured according to γ_d and constraints have been adapted to provide a consistent reduction, the reduced feature model FM' is deployed on all devices compatible with the configuration

γ_d. Although our approach addresses only the reduction of the feature model itself, the reduction may also be used to reduce the deployable software application. To reduce the application the implementation of features which are classified as bound-dead is removed because the implementation is incompatible to the hardware context and would never be used during runtime of the device.

The final result of this brute force reduction process produces a feature model FM' such that each consistent configuration of the original feature model FM that extends the partial configuration γ_d corresponds to a consistent reduced configuration of the stepwise reduced feature model FM' for the following reasons: The reduction process for a given FM and partial configuration above preserves the required property as proven in Theorem 1.

5.2.2 Reducing the Case Study Example

In this section we apply the reduction process to the feature model of the case study in Figure 1. Using the hardware context given in Figure 3 as a partial configuration, we are able to reduce the original feature model to its unbound features. Figure 7 shows the unbound features and the remaining constraints for the hardware context of the smartphone product line. These unbound features require the features provided by the hardware context to be bound-alive on the target device.

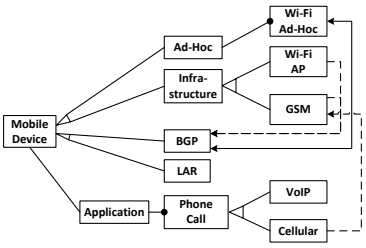


Figure 7: Reduced Feature Model for Figure 1

Using the reduced feature model in Figure 7 we are now able to compute specific configurations according to the dynamically changing requirements of the context more efficiently. Considering our context scenarios in Section 3 a suitable configuration is computed when changing the location from (i) Office Ad-Hoc to (ii) City GSM.

6. EVALUATION

This section provides an evaluation of our feature model reduction approach presented in this paper based on simulation. The evaluation illustrates the benefit of using a reduced feature model for a reconfiguration at runtime instead of the original feature model. Therefore, the evaluation focuses on a comparison of the time to compute a configuration and resources spent during such a computation.

6.1 Simulation Setup

As simulation environment we used a simulation server with 3.4GHz and 8GB of RAM. As the test model we used a feature model FM containing 72 features from the SPLOT[3] repository. We randomly created a configuration γ by binding features to bound-alive (20%) and bound-dead (5%). Reducing the FM according to γ using the Algorithm 1 resulted in a reduced feature model FM' with 32 features (44% smaller). One simulation run consisted of 300 configuration requests on both feature models FM' and FM using a partial configuration γ'. Each (re-)configuration represents a change in the contextual environment and would trigger an adaptation on a device. Therefore, a complete and valid configuration is computed that may be used to reconfigure the device.

In each simulation run we recorded the *time* to compute a complete configuration and the executed *operational steps*, e.g., method calls. The operational steps are used as an abstraction for the consumed computational resources. To validate our results and to identify deviations in our results, we executed 500 simulation runs.

[3]http://www.splot-research.org/

6.2 Results

The results for comparing the execution time for the computation of a complete configuration for γ' in comparison for FM and FM' are depicted in Figure 8(a). During the computation of 300 reconfigurations a single computation was approximate 60% faster in averge if we used the reduced feature model FM' instead of the original feature model FM. The variance in the simulation runs result from the different, randomly generated configuration requirements γ' since they differ in their complexity between 3 to 12 features.

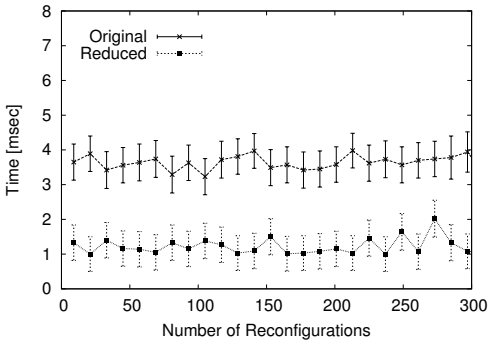
Figure 8(b) shows the amount of executed operations over 300 reconfiguration requests. Again, the computation using a reduced feature model outperforms the computation using the original feature model. During the processing of 300 reconfigurations 66% less operations were executed if a reduced feature model is used. This implies that 34% less computational power was utilized and, thus, solving on a reduced feature model proofs to be more energy efficient. Compared to the runtime comparison, the variance in the amount of executed operations was about four times higher if the original feature model FM was used.

However, reducing a feature model is costly: the reduction of FM using our Algorithm 1 executed 2264×10^3 operations. Computing a configuration executed 17×10^3 or 8×10^3 operations in average, using FM or FM' respectively. This implies, that we have to execute approximate 250 reconfigurations using FM' to regain the resources spend in the reduction of FM. These results shows, that a reduction of complex feature models should be computed on a device that has the sufficient resources, e.g., a deployment server. Nevertheless, for our simulation setup we are up to 60% fester and use up to 34% less computational resources if we use a reduced feature model.

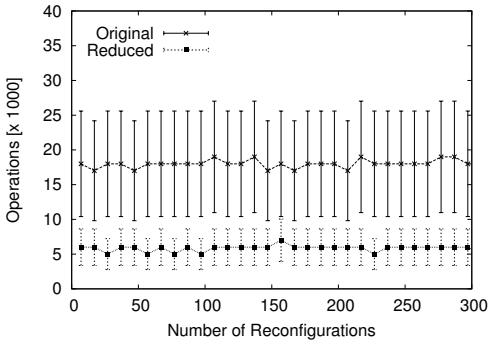
Naturally, our results depend on the amount of reduced features. Our test model was reduced by 44%; if the level of reduction is lower, the difference between solving on an original and reduced feature model are smaller.

7. CONCLUSION AND FUTURE WORK

In this paper we proposed to apply dynamic software product lines to the domain of dynamic adaptive systems. Introducing a context aware case study focusing on mobile devices we derived several contextual categories and challenges. To approach the challenges of resource limitation and heterogeneity in dynamic adaptive systems, we introduced a reduction methodology to minimize a feature model according to the hardware context of a device. We provided



(a) Average time computation



(b) Average number of operations

Figure 8: Comparison computing a Configuration at Runtime with a reduced and original Feature Model

a formal approach to reduce a feature model to a minimal set of variable features by removing features that are bound according to a partial configuration. With this approach, we reduce a feature model on a powerful device, like a server, before the application is installed on a resource limited device, like a smartphone. As a consequence the computation of a configuration at runtime is faster and utilizes fewer resources. Setting up a simulation based on a deployment scenario, we evaluated our reduction approach. The results of the simulation were promising and illustrated how much resources may be saved using a reduced feature model. Additionally, the results lead to the conclusion that amount of improvement depend of the amount of reduced features.

Our next steps in future research focus on three major areas: (i) validate our deploy scenario on a set of mobile devices, (ii) include qualitative characteristics in features and the configuration process, and (iii) adopt a structured approach for modeling and composing multiple contextual categories each inducing reduced pre-configurations and constituting a valid specialization of the feature model, cf., [16].

8. REFERENCES

[1] R. Ali, R. Chitchyan, and P. Giogini. Context for Goal-level Product Line Derivation. *3rd International Workshop on Dynamic Software Product Lines*, pages 24 – 28, 2009.

[2] N. Bencomo, P. Sawyer, G. Blair, and P. Grace. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In *Proceedings of 2nd International DSPL Workshop*, 2008.

[3] J. Bosch. *Design and Use of Software Architectures - Adopting and Evolving a Product Line Approach*. Addison-Wesley, 2000.

[4] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2(5):483 – 502, 2002.

[5] C. Cetina, J. Fons, and V. Pelechano. Applying Software Product Lines to Build Autonomic Pervasive Systems. In *Proceedings of 12th International Conference on SPL*, pages 117 – 126, 2008.

[6] C. Cetina, V. Pelechano, P. Trinidad, and A. R. Cortés. An architectural discussion on dspl. In *Proceedings of 12th International Conference on SPL*, pages 59 – 68, 2008.

[7] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, 2000.

[8] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In R. Nord, editor, *Software Product Lines*, volume 3154 of *LNCS*, pages 162 – 164. Springer, 2004.

[9] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. In *Software Process Improvement and Practice*, 2005.

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

[11] H. Hartmann and T. Trew. Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. *Proceedings of 12th International Conference on SPL*, pages 12 – 21, 2008.

[12] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and S. A. Peterson. Feature Oriented Domain Analysis (FODA). Technical report, Carnegie-Mellon University, 1990.

[13] A. S. Karataş, H. Oğuztüzün, and A. Doğru. Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains. In J. Bosch and J. Lee, editors, *Software Product Lines: Going Beyond*, LNCS, pages 286 – 299. Springer, 2010.

[14] Y.-B. Ko and N. H. Vaidya. Location-aided routing (lar) in mobile ad hoc networks. *Wireless Networks*, 6(4):307 – 321, 2000.

[15] C. Peper and D. Schneider. On runtime service quality models in adaptive ad-hoc systems. In *Proceedings of the Workshop on Software integration and evolution @runtime*, pages 11 – 18, 2009.

[16] J. Schroeter, M. Lochau, and T. Winkelmann. Multi-Perspectives on Feature-Models. In *Proceedings of the 15th International MODELS Conference*. Springer, 2012 (to appear).

[17] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management, The 6th International Conference on Ubiquitous Computing*, 2004.