# Coordinated Run-time Adaptation of Variability-intensive Systems: An Application in Cloud Computing

Andreas Metzger[†], Andreas Bayer[†], Daniel Doyle[*], Amir Molzam Sharifloo[†], Klaus Pohl[†], Florian Wessling[†]
[†]paluno (The Ruhr Institute for Software Technology), University of Duisburg-Essen, Germany
<firstname>.<lastname>@paluno.uni-due.de
[*]Intel, Dublin, Ireland
danielx.doyle@intel.com

## ABSTRACT

Distributed systems, such as cloud systems or cyber-physical systems, involve the orchestration of different variability-intensive, adaptive sub-systems. Each of these sub-systems may perform adaptations simultaneously and independently from each other. Yet, if dependencies between the adaptations of the sub-systems are not considered, this may lead to conflicting adaptations or untapped synergies among adaptations.

This paper introduces FCORE, a model-based approach, which facilitates coordinating adaptations among variability-intensive systems. The permissible run-time reconfigurations of each system is specified by an FCORE model, which combines feature models used in Dynamic Software Product Lines with goal models. FCORE models are mapped to constraint satisfaction problems to determine conflicts and synergies among the adaptations of the systems during execution. We demonstrate the FCORE approach by using a cloud system as a typical exemplar for a distributed system. The cloud system is part of an industrial use case concerned with offering value-added cloud services.

## Keywords

Adaptive systems, dynamic software product lines, variability, cloud

## 1. INTRODUCTION AND MOTIVATION

Adaptive software systems are capable of modifying their structure and behavior at run time in order to cope with dynamic context changes. Such changes include the evolution of used services and their actual quality provided during run-time, sensors available during system operation to obtain environment information, the availability of other systems to interact and cooperate with, as well as the amount and quality of data that is collected [21].

Distributed systems, such as cloud systems or cyber-physical systems, may involve the orchestration of various variability-intensive, adaptive (sub-)systems, each of which may perform adaptations simultaneously and independently from each other. As an example from cloud computing, the adaptation of the cloud infrastructure (IaaS) may happen independently from the adaptation of the cloud applications (SaaS) running on top such infrastructure [8, 12]. The SaaS system may activate additional features due to user demand, whilst the IaaS system may deactivate certain virtual machines (processors) to save energy. This in turns leads to a conflict as not enough IaaS resources are available to run the additional features of the SaaS with sufficient performance.

Consequently, such kinds of dependencies between variabilty-intensive, adaptive systems have to be considered, to avoid conflicting adaptations, but also to exploit potential synergies between complementary adaptations of different systems. This paper introduces FCORE, a model-based approach, which facilitates coordinating adaptations among variability-intensive systems by exploiting solutions from Dynamic Software Product Lines (DSPLs). DSPLs are a well-known approach to realizing run time adaptation, in situations where the possible adaptations can be anticipated and codified during design time [21].

A DSPL extends existing software product line engineering approaches by moving their capabilities to run time [15]. In particular, variability binding is postponed to run time, allowing a DSPL to activate or deactivate certain features according to context changes. In a classical software product line, variability describes different possible products, i.e., software systems. In contrast, DSPL variability describes different possible configurations (i.e., adaptations) of the same system. The configurations of a DSPL are expressed in terms of a product line variability model, usually a feature model [21].

FCORE extends DSPL feature models with goal models in order to support coordination among different DSPL systems. Potential system adaptations are described using concepts from feature models. The run-time reconfigurations of each system is thereby specified by an FCORE model. High-level dependencies between adaptations of different systems, and thus links between the FCORE models, are described using concepts from goal models, in particular softgoals and contribution links.

FCORE aims to find a configuration for each of the different systems that jointly achieves optimal softgoal satisfaction. To this end, FCORE models of the systems are mapped to a constraint satisfaction problem (CSP). The CSP facilitates finding the configurations that lead to optimal goal satisfaction, whilst conforming to the configuration constraints imposed on feature configuration. As an example, the aforementioned adaptations of the IaaS and SaaS cloud services both have a negative impact on response time. FCORE allows modeling this dependency by specifying, say, a joint "performance" softgoal. Optimizing the CSP derived from the FCORE

*VACE '16 Austin, Texas USA*
© 2016 ACM. ISBN 978-1-4503-4176-9/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2897045.2897049

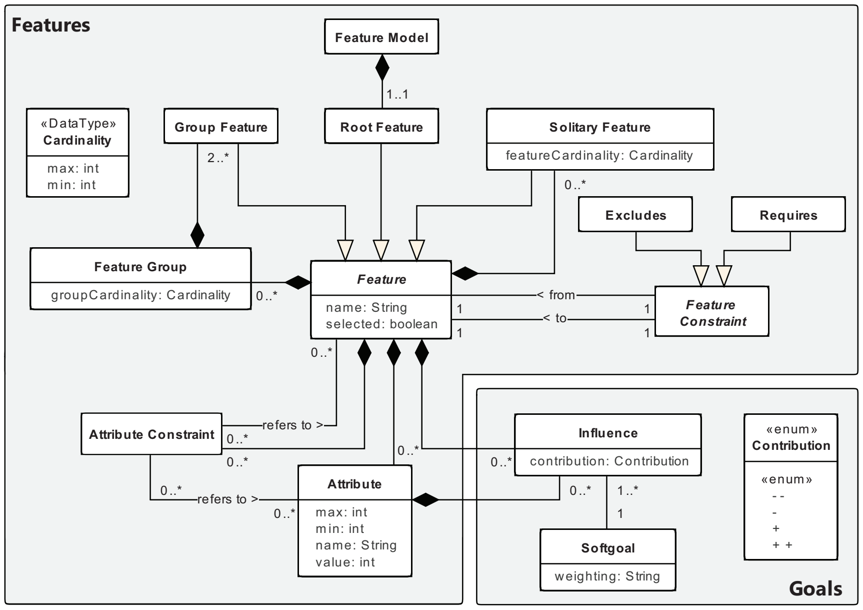


Figure 1: FCORE Metamodel

models will lead to a configuration of both IaaS and SaaS which achieves the minimal response time expected by the users. In our example, more IaaS resources may be added, or the additional SaaS feature may be turned off.

It should be noted that – despite using cloud systems as exemplars – FCORE is not relying on specific characteristics of cloud systems. FCORE is positioned as a more generic approach for coordination among variability-intensive, adaptive systems. Also, it is worth mentioning that FCORE is not aiming to address other coordination concerns, such as synchronization among systems (e.g., see [16]), or oscillations across adaptations (e.g., see [6])

This paper introduces the FCORE approach, which comprises a graphical modeling language, as well as a formalization of the language (mapping to CSP) that facilitates automatic reasoning on FCORE models during runtime. We demonstrate the applicability and use of FCORE in a cloud case study. Concrete FCORE models are described, the efficacy of FCORE is illustrated, and the performance of FCORE is analyzed.

The remainder of the paper is structured as follows. Section 2 introduces the FCORE modeling language and its formalization. Section 3 describes the use of FCORE in the cloud computing case study. Section 4 positions FCORE with respect to existing work in the field. Section 5 concludes and provides an outlook.

## 2. FCORE

We introduce the main modeling concepts of the FCORE language and then describe the formalization of the FCORE concepts.

### 2.1 FCORE Modeling Language

Fig. 1 shows the UML metamodel of the FCORE language, which is structured into two parts: *Features* and *Goals*.

**Features:** The *Features* part uses concepts from extended feature models as most expressive starting point to model adaptations (a justification for the use of extended feature models is provided in Section 4.1). Each combination of features described by the feature model represents a valid system configuration and thus adaptation of the system. Attributes allow modeling numerical properties and defining additional feature constraints (e.g., see [2]).

Following [10], the modeling of adaptations starts from the concept of `Feature Model` which consists of a `Root Feature`. The Root Feature consists of multiple sub-features resulting in a tree-like structure. These subfeatures are `Solitary Features` with a `featureCardinality` or `Group Features` in a `Feature Group`. A feature group defines permissible combinations of sub-features. The use of feature cardinalities allows the instantiation or cloning of features to reduce redundancy [11]. Identical feature subtrees have to be modelled only once and can be instantiated multiple times at run time. Two special cases of feature cardinalities are mandatory (1..1) and optional (0..1).

A feature may have `Attributes` which are properties of a feature. Attributes contain an integer `value` between `min` and `max`. A feature can only be selected if all of its `Attribute Constraints` are fulfilled [17, 20]. Attribute constraints define relations between attributes and feature selection dependencies [2].

**Goals:** The *Goals* part includes concepts from goal modeling, in particular softgoals and contribution links (e.g., see [29]). Shared softgoals between different systems serve as high-level specification of potential interactions between adaptations.

Features and attributes are related to `Softgoals` by using `Influences` relationships (aka. contribution links). We suggest discretizing the impact of a feature or attribute on a softgoal in terms of four different degrees. The value “$++$” means a softgoal is completely fulfilled if the corresponding feature is selected and not fulfilled for “$--$”. A generally positive influence is described by “$+$” and a negative by “$-$”.

For features with cardinalities all influence relations will be cloned accordingly. This allows taking into account the number of feature instances when calculating the softgoal value. As mentioned above, for coordinated adaptation, shared softgoals are used to combine the different FCORE models of the cloud entities.

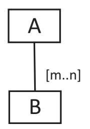
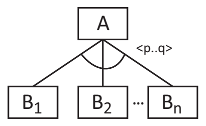
### 2.2 FCORE Formalization

To facilitate coordination, we provide a formalization of FCORE models as a Constraint Satisfaction Problem (CSP). CSPs allow automatic reasoning on FCORE models during runtime, by exploiting existing CSP solvers. Due to the use of extended feature models (and thus the use of attributes with integer domains), we chose to map the FCORE models to a Constraint Satisfaction Problem (CSP). CSP solvers natively handle variables with numerical domains in contrast to SAT solvers which use only Boolean variables and thus would require an encoding of attributes.

The concepts of the `Feature` part of the FCORE models are formalized as constraints in the CSP. The concepts of the `Goal` part are formalized as a "utility" function that is to be optimized as part of the CSP; e.g., finding the highest softgoal fulfillment among various configurations among systems, whilst conforming to the configuration constraints imposed on feature configuration.

**Features:** We follow the approach from product line engineering to represent a feature "A" as a variable $A \in \{0,1\}$, with 1 meaning the feature is part of the configuration, and 0 otherwise. The formalization of feature cardinalities and feature groups is shown in Table 1.

Table 1: Formalization of Feature Cardinalities and Groups

| | |
|---|---|
|  | if $(A = 0)$<br>$\sum_{x=1}^{n} B_x = 0$<br>else<br>$m \leq \sum_{x=1}^{n} B_x \leq n$ |
|  | if $(A = 0)$<br>$\sum_{x=1}^{n} B_x = 0$<br>else<br>$p \leq \sum_{x=1}^{n} B_x \leq q$ |

In case of feature cardinalities, each feature instance $x$ of "B" is represented by a variable $B_x \in \{0,1\}$, where 1 means the instance exists and 0 otherwise (cf. [20]). The first row in Table 1 shows an example for feature "A" having a subfeature "B" with feature cardinality $[m..n]$. This formalization differs from the work in [4], which uses only a single variable for representing feature cardinalities. Yet, to differentiate between feature instances with different selected subfeatures and attribute values it is necessary to introduce a variable for each possible feature instance.

The formalization of feature groups is similar to feature cardinalities (see the second row in Table 1). In this case subfeatures do not represent cloned instances but distinct features of which a minimum of $p$ and a maximum of $q$ may be selected[1].

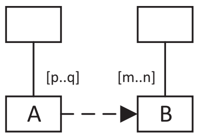
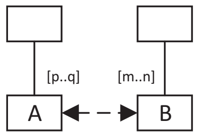
Table 2 shows the formalization of requires and excludes relations, which takes particular account of feature cardinalities as formalized above.

Feature attributes are formalized as variables $val \in \{min..max\} \cup \{0\}$ with a minimum and maximum value ($min > 0$, $max > 0$). The value is set to zero if the corresponding feature "A" is not selected. This is necessary for summing attributes and calculating softgoal values (e.g., see the similar approach in [26]). A feature can only be selected if all of its `Attribute Constraints` are fulfilled [17, 20]. Attribute constraints define relations between attributes and feature selection dependencies [2].

**Goals:** The main elements of goal formalization are shown in Table 3. The value of a softgoal is expressed as a float variable $sgVal \in [0.0, 1.0]$, with 0.0 meaning non-satisfaction and 1.0 meaning full goal satisfaction. $sgVal$ serves as "utility" function seeking

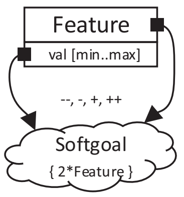
[1]OR groups with $< 1..n >$ and XOR groups with $< 1..1 >$ are two special cases of feature groups [2].

Table 2: Formalization of Requires and Excludes with Cardinalities

| | |
|---|---|
|  | if $(\sum_{x=1}^{n} B_x = 0)$<br>$\sum_{x=1}^{q} A_x = 0$ |
|  | if $(\sum_{x=1}^{n} B_x > 0)$<br>$\sum_{x=1}^{q} A_x = 0$ |

to be optimized when solving the CSP (also see Section 3.2). Due to the normalization of goal satisfaction to the range $[0.0, 1.0]$, feature and attribute variables have to be normalized to that interval as well (see $v'_x$ in Table 3). Depending on the influence relation (i.e. "$--$", "$-$", "$+$" or "$++$") the attribute or feature has a different impact on the overall softgoal value. A feature with a "$++$"-influence yields a softgoal value of 1.0 if selected, and 0.0 otherwise (vice versa for "$--$"). An attribute with "$+$"-influence will move the overall softgoal value closer to 1.0 the bigger and closer to 0.0 the smaller its value is (vice versa for "$-$"-influence).

Table 3: Formalization of Goals

| | | |
|---|---|---|
|  Feature<br>val [min..max]<br>--, -, +, ++<br>Softgoal<br>{ 2*Feature } | $contribution =$ "$+$":<br>$v'_x := \frac{v_x - min + 1}{max - min + 1}$<br>$v'_x := 0$, if $Feature = 0$ | $contribution =$ "$-$":<br>$v'_x := 1 - \frac{v_x - min + 1}{max - min + 1}$<br>$v'_x := 1$, if $Feature = 0$ |
| | $sgVal := \frac{k_1 \cdot v'_1 + k_2 \cdot v'_2 + \ldots + k_n \cdot v'_n}{\sum_{x=1}^{n}(k_x)}$ | |

For features with cardinalities all influence relations will be cloned accordingly. This allows taking into account the number of feature instances when calculating the softgoal value.

## 3. CLOUD USE CASE

Cloud environments are made up of many different entities [14]. Those include, to name only a few, virtual resources deployed on top of physical resources and offered as Infrastructure-as-a-Service (*IaaS*), web or application servers deployed inside virtual machines, as well as application software deployed on top of web or application servers and offered as Software-as-a-Service (*SaaS*). In principle, each of these cloud entities may perform various different adaptations at run time. Examples include virtual machine migration, load balancing or dynamic service deployment.

Currently, cloud entities take adaptation decisions simultaneously but often independently from each other [8]. However, there are many complex dependencies between the entities in a cloud environment. For instance, the performance of a SaaS may depend on how many and which kinds of virtual machines (IaaS) are employed [19]. For any given situation, different concrete adaptations may be applicable – often simultaneously for the different cloud

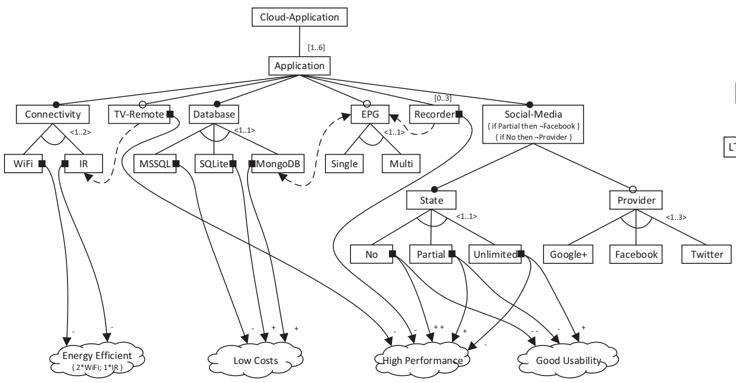


Figure 2: FCORE Model of Cloud SaaS

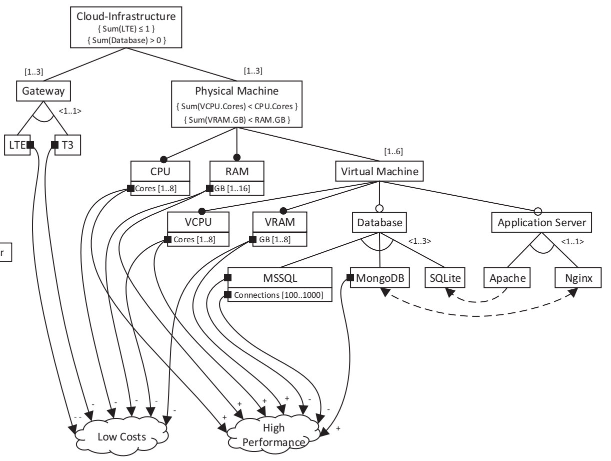


Figure 3: FCORE Model of Cloud IaaS

entities. Combining contradictory adaptations or not considering potential synergies among adaptations may negatively affect quality of service and cost [6]. Coordinated run-time adaptation thus is of particular importance in cloud computing.

## 3.1 FCORE Models for Cloud Use Case

We first demonstrate how FCORE can be applied to model the variability of cloud systems and the dependencies between these systems in terms of adaptation. We use an industry use case from the EU FP7 project *CloudWave* [5]. The use case involves a SaaS and IaaS system developed and operated by an industry member of the project.

The SaaS, accessible through a mobile phone App as a front-end, offers value-added services to accompany running TV programmes. Features offered by the SaaS include providing additional information (such as digital TV guide and live voting), as well as connection with various social networks. The current TV programme has strong influence on the number of user requests; e.g., such as during sporting events. Therefore, the SaaS is hosted on an elastic cloud infrastructure (IaaS) to ensure quality of service in the presence of fluctuating demand.

Four main kinds of adaptations occur in the cloud use case:

- *IaaS Vertical Scaling*, i.e., change of the resources assigned to a virtual machine, such as CPU cores and RAM size.
- *IaaS Horizontal Scaling*, i.e., adding/removing virtual machines to distribute load across more/less virtual machines.
- *IaaS (Re-)Deployment of Servers*; e.g., databases, web servers and application servers.
- *SaaS Changing Service Offerings*; e.g., only offering the social networking feature of the application to paying customers in case of very high demand.

The variability of the aforementioned SaaS and IaaS systems and their softgoals have been modeled with FCORE, and models are shown in Fig. 2 and 3 respectively. Examples for FCORE modeling concepts used in these two models include the following.

**Features:** In the SaaS model, for the *Social-Media* feature, only one of its sub-features may be chosen, which is expressed through a feature group constraint. Another example is the use of the feature cardinality of 1..6 for the *Application* feature, which means that an App has at least one and up to six instances. In the IaaS example, for instance, the *Database* feature is modeled as optional.

The IaaS model includes, as example, the attribute *CPU.Cores*, which defines the number of CPU cores which can range from 1 to 4. The constraint *Sum(Database)*$>$ 0 (defined as part of the IaaS root feature) means that at least *Database* instances among all *Virtual Machine* instances has to be selected.

**Goals:** Both FCORE models share the common softgoals *High Performance* and *Low Costs*. This means that dependencies and thus the need for coordination among the adaptations of the IaaS and SaaS systems is expressed by means of these softgoals. As see in the models, the attributes *CPU.Cores* as well as *RAM.GB* of the IaaS system have a positive influence on the softgoal *High Performance*, while the feature *Social Media: Unlimited* of the SaaS has negative influence on the softgoal *High Performance*.

It should be noted that in practice, both cloud systems and thus their respective FCORE models models are typically developed by different actors, such as App developers vs. infrastructure owners. FCORE models may be developed independently from each other, provided that a common understanding of shared softgoals has been established. Establishing such common understanding is supported by the high-level abstraction expressed in goal models (cf. Section 4.3).

## 3.2 Efficacy of FCORE

To illustrate the efficacy of FCORE, we explain how FCORE models may be used to drive adaptation decisions as part of a cloud stack. We describe the high-level architecture of the cloud stack developed in the *CloudWave* project, which aims to extend OpenStack with advanced DevOps capabilities, such as coordinated adaptation and feedback-driven development [5].

The concept of coordinated adaptation plays a pivotal role in the architecture of CloudWave and is implemented in the *Adaptation Engine*. The Adaptation Engine is comprised of a *message queue* (Listener/Publisher), adaptation *distributors* and *consolidators*, and a *plugin* API as shown in Fig. 5.

When an adaptation request is received (e.g., because of an observed performance violation), the Adaptation Engine uses a series of *Plug-Ins* to create, filter, validate and ultimately decide upon

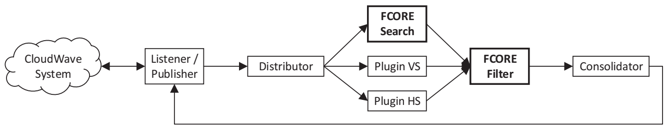
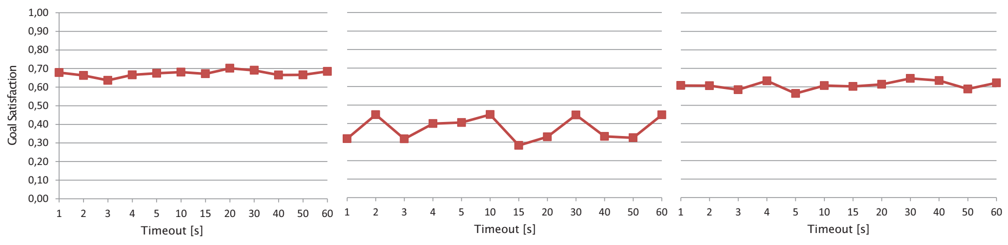


Figure 5: CloudWave Adaptation Engine and Roles of FCORE



(a) SaaS model (b) IaaS model (c) Combined model

| Model | Features | Variables | Avg. Goal Satisf. |
|---|---|---|---|
| SaaS | 23 | 133 | 0.67 |
| IaaS | 17 | 259 | 0.37 |
| Combined | 40 | 392 | 0.61 |

(d) Statistics

Figure 4: Performance of FCORE Search

concrete adaptation actions. Each Plug-In may address specific optimization concerns and work on different knowledge. The *Distributor* accepts a list of potential adaptation requests and manages the initialization and execution of the appropriate plug-in processes to handle these requests. The *Consolidator* accepts the output of one or more plug-ins, and works to combine the output lists into a single list of actions.

FCORE may be incorporated into the *CloudWave* Adaptation Engine as follows.

**FCORE Search** aims at finding an adaptation applicable in a given context situation. In our use case, the cloud system may face such high workload that adding more IaaS resources (e.g., CPU Cores) alone will not be enough. As we have seen above, the kinds of SaaS *Social-Media* features also have an impact on performance. In order to find the set of coordinated adaptations among IaaS and SaaS that may be able to handle this high workload, the FCORE models are thus analyzed to find a configuration with a higher satisfaction of the softgoal *High Performance*. Technically, this means that *FCORE Search* receives an adaptation request from the *Distributor*. Then, *FCORE Search* aims to solve the CSP (which combines the SaaS and IaaS models) to find configurations that meet constraints and achieve better goal satisfaction than the initial configuration. As a result, in our example FCORE may suggest the configuration of *No Social Media* for the SaaS and *6 CPU Cores* for IaaS.

**FCORE Filter** takes configurations provided by one or more of the plug-ins and checks the configurations for consistency with the CSP. As shown in Fig. 5, such plugins may perform vertical scaling (VS) or horizontal scaling (HS). *FCORE Filter* thereby can identify HS and VS adaptations that negatively impact on softgoal fulfillment and thus should not be enacted. Thereby, *FCORE Filter* can eliminate horizontal or vertical scaling actions that may decrease softgoal fulfillment.

## 3.3 Performance Assessment of FCORE

The applicability of FCORE for coordinating cloud adaptation depends on the time that is required to find a “good” solution to the CSP. Adaptation decisions need to be taken fast enough such as to timely respond to changes to be useful. Based on a given time available for taking an adaptation decision, the CSP solution delivered within this given time-frame should thus be sufficiently close to an optimal solution.

Several performance aspects for feature model analysis have been covered in the literature (e.g., [3, 23, 30]). We wanted to gain insights concerning the influence of the specific FCORE concepts (such as feature cardinalities, influence relations and softgoals) on performance. We thus performed two initial experiments using the cloud use case models: (1) measuring the time required to check for validity of a configuration (*FCORE Filter*), and (2) assessing the dependency of goal fulfillment on the time available for CSP solving (*FCORE Search*).

The experiments were performed using the JaCoP[2] Constraint Logic Programming Library 4.2.0, compiled and launched with Eclipse Luna SR 1a Version 4.4.1 and OS X Yosemite 10.10.3 running on an Apple MacBook Pro with 2,6 GHz Intel Core i7 CPU and 16 GB DDR3 RAM. The FCORE models contain 23 and 17 features respectively, and their formalization involved 133 and 259 CSP solver variables respectively.

**FCORE Filter:** FCORE Filter performs a validity check of a given configuration. To this end, we examine a given feature selection for constraint satisfaction (similar to [3]). We have performed measurements for the different models, which resulted in an average 2*ms* for checking the validity of a given configuration with respect to its FCORE model.

**FCORE Search:** FCORE Search aims to find a solution to the CSP where the feature configuration leads to high goal satisfaction. We have compared the performance of computing goal fulfillment for the SaaS and IaaS models in isolation, and also – more importantly – for the combined models.

We use the softgoal values as part of a utility function $f = \mathit{High\ performance} \cdot \mathit{Low\ Cost}$ and instruct the CSP solver JaCoP to find a valid configuration with the highest softgoal fulfillment. We use JaCoP’s `IndomainRandom` solution space traversal strategy to eliminate a possible influence of the model structures on softgoal fulfillment.

The goal fulfillment with respect to the time that was given the JaCoP for solving are shown in Fig. 4. For the SaaS model (Fig. 4a), goal fulfillment levels vary slightly between 0.64 and 0.71. For the IaaS model (Fig. 4b), goal satisfaction in general is lower and satisfaction levels show a higher variation between 0.28 and 0.45. This can be attributed to the use of feature attributes, which require a large domain of values to be explored. Additionally, 259 solver variables were used (almost twice number of variables compared to the SaaS model). Finally, for the combined model (Fig. 4c), in comparison with the IaaS model, the combined model has a higher goal fulfillment on average. This observation can be explained by the amount of influence relations which is higher for the combined model. The variation of goal fulfillment due to attributes with their bigger range from infrastructure is balanced by the influence relations between features and goals from the application.

Overall, the current results indicate that goal fulfillment does not significantly improve if more time were given to the CSP solver. And thus, good FCORE Search results can already be achieved with short response time (below a second).

[2]http://jacop.osolpro.com

## 4. RELATED WORK

Adaptive systems have emerged as a broad research area over the last decade. We focus on key relevant work on (i) dynamic product lines, (ii) goal-based adaptation, and (iii) cloud adaptation.

## 4.1 Dynamic Software Product Lines

The feature model of a DSPL describes different possible adaptations of the same system. Several authors propose using basic feature models (FMs) for DSPLs (e.g., [1]). While basic FMs allow efficient analysis (e.g., using SAT solvers), basic FMs may become bloated quickly and thus may offer limited understandability to developers. For example, modeling of configuration "ranges" is a problem. In addition, modeling multiple instances of components means modeling them explicitly using multiple sub-features. To address some of these shortcomings, other authors advocate using cardinality-based feature models (e.g., [22]). These offer UML-like cardinalities for feature selection [$m..n$]. Cardinalities facilitate the cloning of a feature together with its sub-tree [11]. Still, cardinality-based FMS do not provide a concise way of modeling "ranges". As a further extension of FMs, extended feature models thus have been proposed for usage in DSPLs [28]. Extended FMs add numeric feature attributes; e.g., such as over integer domains. This facilitates modeling feature ranges in a compact and concise form. Extended FMs appear best suited for design time modeling of adaptive systems, however at the price of more difficult analysis (extended FMs typically require CSP solvers).

Despite their ability to model adaptations of a single system, existing DSPL approaches have not been developed to handle interactions and dependencies between adaptations of multiple systems. For context-adaptive systems there are DSPL proposals that map changes in the system context to adaptations expressed in feature models [1, 18]. However, this only facilitates interactions in one direction, i.e., from the context to the system. For coordinated adaptation, interactions need to be captured in both directions due to mutual dependencies between adaptive systems. FCORE addresses this gap.

## 4.2 Goal-Based Adaptation

FCORE extends feature models with the concepts of softgoals to facilitate coordination among adaptations of different cloud entities. Other authors have also proposed relating goals to features in order to support adaptation. Most importantly [9, 24, 27] exploit impact links from hardgoals (expressing the software architecture) to softgoals (expressing the requirements). In a sense, a tree of hardgoals is similar to a basic feature model. However, their intention, is not to facilitate *coordinating* different adaptive systems but to facilitate the combination of requirements adaptation (goals) and architecture adaptation (features) within a *single* system.

## 4.3 Cloud Adaptation

Most adaptation solutions for cloud environments focus on cloud systems in isolation. For example, [13] focus on the infrastructure and [7] focus on applications. With few exceptions, most solutions do not consider the possible interferences between cloud adaptations. In previous work, we have elaborated the problem of coordinated cloud adaptation and provided an analysis of the involved challenges [8].

The SALOON framework by Quiton et al. [25] uses feature models with cardinalities, attributes and constraints to support automated configuration among federated cloud infrastructures. Even though they consider dependencies among different IaaS systems (federated cloud), their work addresses deployment time and not run-time adaptation.

The framework presented by De Oliveira et al. [12] is closest to our work. They focus on coordination between two distinct kinds of adaptation managers: one for the application and one for the infrastructure. Each adaptation manager is able to make decisions locally but may also communicate with its counterpart by exchanging adaptation actions or accessing a shared knowledge base. To ensure coordination, critical sections in the knowledge base are defined which may not be modified concurrently. In addition a protocol is defined for exchanging adaptation actions. The coordination and synchronisation is addressed at the level of concrete adaptation actions and information elements. Those actions and elements need to be mutually agreed between cloud developers. In contrast, we address coordination at a high level of abstraction by introducing goals to synchronize adaptations.

## 5. CONCLUSIONS AND PERSPECTIVES

This paper introduced FCORE, a model-based approach for coordinated adaptation of variability-intensive systems. FCORE extends feature modeling languages and reasoning techniques from dynamic software product lines. Using softgoals as a high-level abstraction to model potential interactions and dependencies between adaptations, FCORE provides a loose coupling between adaptive systems to detect adaptation conflicts or missing synergies.

An initial evaluation of FCORE has indicated the efficiency of using FCORE models for coordinating adaptations. Even though our initial evaluation results appear promising, we are aware that further empirical studies and experiments are required. As part of ongoing work, we are evaluating the measurable improvements that FCORE delivers as part of the CloudWave cloud stack.

**Acknowledgments:** We cordially acknowledge the constructive comments provided by the anonymous reviewers. Research leading to these results has received funding from the EU's 7th Framework Programme under grant agreement 610802 (CloudWave).

## 6. REFERENCES

[1] G. H. Alferez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software*, 91:24–47, 2014.

[2] D. Benavides, P. T. Martin-Arroyo, and A. R. Cortes. Automated Reasoning on Feature Models. In O. Pastor and J. F. e. Cunha, editors, *CAiSE 2005, Porto, Portugal, June 13-17*, volume 3520 of *LNCS*, pages 491–503, 2005.

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

[4] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortes. Using Java CSP Solvers in the Automated Analyses of Feature Models. In *Generative and Transformational Techniques in Softw. Eng.*, LNCS, pages 399–408. 2006.

[5] Bruneo, D. et al. CloudWave: Where adaptive cloud management meets DevOps. In *IEEE ISCC 2014 Workshop on Management of Cloud Systems (MoCS 2014), June 23rd, 2014, Madeira, Portugal*, 2014.

[6] A. Bucchiarone, C. Cappiello, E. Di Nitto, S. Gorlatch, D. Mailänder, and A. Metzger. Design for self-adaptation in service-oriented systems in the cloud. In D. Petcu and J. Vásquez-Poletti, editors, *European Research Activities in Cloud Computing*, 2012.

[7] A. Bucchiarone, A. Marconi, C. A. Mezzina, M. Pistore, and H. Raik. On-the-Fly Adaptation of Dynamic Service-Based Systems: Incrementality, Reduction and Reuse. In S. Basu,

C. Pautasso, L. Zhang, and X. Fu, editors, *Service-Oriented Computing*, LNCS, pages 146–161. 2013.

[8] C. Cassales Marquezan, F. Wessling, A. Metzger, K. Pohl, C. Woods, and K. Wallbom. Towards exploiting the full adaptation potential of cloud applications. In *ICSE 2014 Workshop on Principles of Engineering Service-oriented Systems (PESOS), May 31, Hyderabad, India*, 2014.

[9] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao. Self-adaptation through incremental generative model transformations at runtime. In P. Jalote, L. C. Briand, and A. v. d. Hoek, editors, *ICSE 2014, Hyderabad, India - May 31 - June 07*, pages 676–687, 2014.

[10] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged Configuration Using Feature Models. In D. Hutchison and et al., editors, *Software Product Lines*, pages 266–283. Berlin, Heidelberg, 2004.

[11] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, pages 7–29, 2005.

[12] F. de Oliveira, T. Ledoux, and R. Sharrock. A Framework for the Coordination of Multiple Autonomic Managers in Cloud Environments. In *SASO 2013*, pages 179–188, 2013.

[13] B. Dougherty, J. White, and D. C. Schmidt. Model-driven Auto-scaling of Green Cloud Computing Infrastructure. *Future Gener. Comput. Syst.*, 28(2):371–378, 2012.

[14] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong. Principles of elastic processes. *IEEE Internet Computing*, 15(5):66–71, 2011.

[15] M. Hinchey, S. Park, and K. Schmid. Building dynamic software product lines. *IEEE Computer*, pages 22–26, 2012.

[16] T. Holvoet and M. Viroli, editors. *Int'l Conference on Coordination Models and Languages COORDINATION, Grenoble, France*, volume 9037 of *Lecture Notes in Computer Science*. Springer, 2015.

[17] A. S. Karatas, H. Oguztuzun, and A. Dogru. 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. 2010.

[18] J. Lee, G. Kotonya, and D. Robinson. Engineering Service-Based Dynamic Software Product Lines. *IEEE Computer*, pages 49–55, 2012.

[19] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas. Service isolation vs. consolidation: Implications for iaas cloud application deployment. In *IC2E 2013, San Francisco, CA, USA, March 25-27*, pages 21–30, 2013.

[20] R. Mazo, C. Salinesi, D. Diaz, and A. Lora-Michiels. Transforming Attribute and Clone-Enabled Feature Models Into Constraint Programs Over Finite Domains. In *ENASE 2011*, Beijing, China, 2011.

[21] A. Metzger and K. Pohl. Software product line engineering and variability management: Achievements and challenges. In J. D. Herbsleb and M. B. Dwyer, editors, *ICSE 2014 (FOSE), Hyderabad, India, May 31 - June 7*, pages 70–84, 2014.

[22] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. Models@ Run.time to Support Dynamic Adaptation. *IEEE Computer*, pages 44–51, 2009.

[23] R. Pohl, V. Stricker, and K. Pohl. Measuring the structural complexity of feature models. In *ASE 2013, Silicon Valley, CA, USA, November 11-15*, pages 454–464, 2013.

[24] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao. Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation. In *RE 2014, Karlskrona, Sweden*, pages 113–122, 2014.

[25] C. Quinton, D. Romero, and L. Duchien. Automated Selection and Configuration of Cloud Environments Using Software Product Lines Principles. In *CLOUD 2014*, Anchorage, United States, 2014.

[26] F. Roos-Frantz, D. Benavides, and A. Ruiz-Cortes. Feature Model to Orthogonal Variability Model Transformations. A First Step. In *JISBD 2009*, pages 81–90, 2009.

[27] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes. Using Constraint Programming to Manage Configurations in Self-Adaptive Systems. *IEEE Computer*, pages 56–63, 2012.

[28] L. Shen, X. Peng, and W. Zhao. Software Product Line Engineering for Developing Self-Adaptive Systems: Towards the Domain Requirements. In X. Bai and et al., editors, *COMPSAC 2012, Izmir, Turkey, July 16-20*, pages 289–296, 2012.

[29] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *5th Int'l Symposium on Requirements Engineering*, pages 249–262, 2001.

[30] J. White, D. C. Schmidt, D. Benavides, P. Trinidad, and A. R. Cortes. Automated Diagnosis of Product-Line Configuration Errors in Feature Models. In *SPLC 2008, Limerick, Ireland, September 8-12*, pages 225–234, 2008.