A Dynamic Software Product Line Approach for Optimal Adaptation of Context-Aware Topology Control Systems

Bachelor Thesis submitted by
Michael Matthé
September 14, 2017

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Real-Time Systems Lab

Department of Electrical Engineering
and Information Technology (FB18)

Adjunct Member Department of
Computer Science (FB20)

Prof. Dr. rer. nat. A. Schürr
Merckstraße 25
64283 Darmstadt

www.es.tu-darmstadt.de

Supervisor: Prof. Dr. rer. nat. Andy Schürr
Advisor: Markus Weckesser

ES-B-0130

Declaration of Authorship for the Bachelor Thesis

I warrant that the Bachelor Thesis presented here is my original work, and that I did not receive any external assistance. All references and other sources I used have been appropriately acknowledged. I further declare that the work has not been submitted for the purpose of academic examination anywhere else, either in its original or similar form.

I hereby grant the Real-Time Systems Lab the right to publish, reproduce, and distribute my work.

Darmstadt, September 14, 2017

(Michael Matthé)

Abstract

Nowadays, components of distributed communication systems adapt themselves continuously to permanently changing environmental contexts, with the aim of achieving certain quality criteria. A comprehensive methodology for specifying runtime adaptability of individual components is provided by dynamic software product lines (DSPL). DSPLs offer methods for developing a set of similar software systems sharing common features and dynamically adapting the system configuration after deployment. However, current approaches do not allow the representation of non-functional properties for the derivation of configuration optimization at runtime in regard to these quality criteria. To this end, we propose a DSPL approach for optimal adaption of context-aware topology control systems. The proposed approach includes automatic feature interaction detection, the quantification of the discovered interactions, and the definition of these interactions as soft-constraints as a mathematical optimization model. The final optimized model contains all the information necessary for optimal runtime adaption in regard to a non-functional property. We evaluated the approach by using two sets of simulation data and one set of synthetic data. The evaluation shows promising results in regard to the correctness of the results in regard to valid configuration as wells as the real world applicability of the optimization model.

Contents

List of Figures

1 Introduction

Communication systems employ topology control to guarantee a high quality of service. The topology control manages the way a network topology is constructed and maintained through the utilization of algorithms. In a communication network the system environment, called context, is constantly changing, which requires the topology control to adapt to maintain the networks optimal functionality. The active topology control mechanisms are determined by the selected system configuration. The communication network systems used in this thesis are developed as part of a software product line. Software product lines are used to design a group of similar software systems which share a common set of features. Due to the changing context it becomes necessary to dynamically adjust the configurations of a system to optimize it in regard to certain non-functional properties. Dynamic software product lines provide a method of developing a common set of features as part of a software system, while expanding the previous approach by also allowing the dynamic reconfiguration of these features at runtime. Software product lines employ feature models to represent the set of possible configurations and confine the given set using constraints, to a set of valid configurations. In a communication network topology this set of features includes features defining the system's behavior, as well as features describing the context given by the current environment.

To optimize a system in regard to non-functional properties the configuration with optimal influence on the given non-functional property has to be chosen. In a given system certain features can interact with each other, creating positive or negative influences on the measured non-functional property. This interaction is more specifically called feature interaction. By finding these feature interactions between system and context features, and evaluating their corresponding influence on a non-functional property, an optimal configuration for the system can be determined for each context. In this thesis we propose finding and using feature interactions from the measurements of a communication network topology in order to encode them as soft constraints for the optimization of the DSPL. We propose the following approach for the optimal runtime adaptions of DSPLs:

- Determine feature-interactions between system features and context features by developing an automatic feature interaction detection using simulation data.
- Quantify the impact of a feature interaction on a non-functional property, in order to compare them quantitatively.
- The major contribution this thesis will make to the approach is the formulation of feature interactions as soft-constraints representing context-dependent features in regard to non-functional properties.

2 Background

2.1 Network Topology and Topology Control for optimizing network metrics

In communication systems topology describes the state of the system as an attributed graph, made up of nodes and links. The graph is simple, without loops or parallel links. Every link has a source and a target node. An schematic network topology is shows in figure 2.1. A path from one node n_a to another node n_z is a list of links from n_a to n_z. In 2.1 one path from n_a to n_c would be given by $\mathrm{P}(n_a, n_c) = (e_{ab}, e_{bc})$. Every node has a set of properties as shown for n_a in 2.1, an identifier ($\mathrm{id}(n_a) = \mathrm{a}$), the position of the node (latitude and longitude) and the hop count, the shortest number of links between the node and another reference node (n_0). The links of a graph, labeled with letter e (edge) and the source and destination node identifiers, have certain properties as well, weight, angle and state (active, inactive, unclassified). [KSV+17]

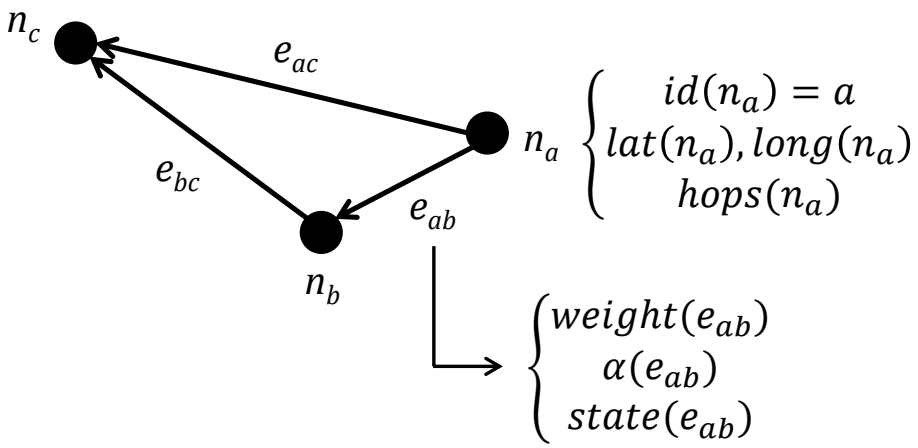

Figure 2.1: Exemplary Network Topology

In order to optimize certain network metrics network topology can be adapted by utilizing topology control (TC). The TC process is divided into three phases: topology monitoring, planning and execution. Topology monitoring detects context events, external modifications of physical topologies which include node and link addition and removal as well as modification of their properties. In the planning phase the TC algorithm analyzes the input topology and produces an output topology, containing all links from the input which are necessary to fulfill consistency properties. The execution phase ensures that all links in the output topology are available for message transfer. The execution can be triggered either periodically or on demand, for example when a batch of context events is finished. [KSV+17]

TC algorithms can be divided into two categories, batch TC algorithms and incremental TC algorithms. Batch TC algorithms analyze the whole input topology and generate a complete output topology based on the analysis. Incremental TC algorithms only analyze the modified portion of the input topology and update the output topology based on the changes. This requires that all links modified since the last increment have to be marked to diffentiate them from unchanged links. [KSV+17]

All TC algorithms must ensure that all links of the output topology are classified and that the graph is connected. An example of a TC algorithm is kTC. When using kTC an output topology link is inactive only if the link is the weight-maximal link in a triangle and the link's weight is k times larger than that of the weight-minimal link in the same triangle. The transmission power required to send a message over a link grows at least quadratically with link length. Therefore kTC preferably adapts the topology to utilize multiple shorter links and remove longer ones based on the previously defined rule set. Therefore kTC would be a desirable TC algorithm to utilize if minimizing energy consumption is demanded. Different TC algorithms and parameters can optimize different quality metrics of the topology such as energy consumption, latency or link lifetime. Depending on what metric needs to be optimized a different TC algorithm must be employed in order to adapt the topology in the optimal way for said metric. [KSV+17]

2.2 Dynamic Software Product Lines

In the past software was relatively small and thus for each variant of a product a new software variant was developed. Software was a simple way to implement different variants of a product that required different hardware variants before. It was easy and cheap to copy, transport and replace. Therefore it was usually not regarded as variable and a customer could either receive software including all features one would ever need or commission software specifically produced for a single purpose. [PBL05]

2.2.1 Software Product Line Engineering

Today almost all systems of a certain complexity contain a substantial amount of software. Product line engineering in software is being adopted for application development since variability can be implemented easier in software than in pure hardware. In [PBL05] Software product line engineering is defined as a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization. [PBL05] defines a platform as a set of software subsystems and interfaces that form a common structure from which a set of derivative products can be efficiently developed and produced. Using these platforms for the development of applications requires the building of reusable parts and reusing these parts. To be able to accomplish adaptions in a reproducible manner, the set of possibilities for adaptions has to be restricted, in order to ensure that the original structure stays intact and comprehensibility and maintainability is preserved. [PBL05]

In a large and complex software system it becomes imperative to clearly define all characteristics and capabilities within that domain to properly allow the reuse of that software as part of an SPL. Domain analysis provides a systematic approach for specifying the characteristics and capabilities of a software system. It creates a common understanding of the domain, thereby improving reuse of the software and enhancing the development process. A domain is defined as a set of current and future applications which share a set of common capabilities and data in [Pet90]. The domain analysis process is divided into three basic phases. The first phase is context analysis which establishes the bounds of the domain as well as the scope for the analysis. In the second phase, domain modeling, the

domain analyst uses the previously defined characteristics, as well as other information sources, to create a domain model, including a feature model. In the third and final phase, the architecture modelling, the structure of software implementations in the domain is the defined, providing developers with a set of architectural models to use for construction of the applications. [Pet90]

2.2.2 Feature Models

In SPLs feature analysis is utilized for modelling the general capabilities of an application and to communicate its requirements to the end user. A *feature* is defined in [Pet90] as a prominent or distinctive user-visible aspect, quality or characteristic of a software system. Feature models provide a way for specifying commonality as well as variability for products of SPLs. They consist of an application's standard features and the relationship between those features. [Pet90]

In figure 2.2 an example of a feature model is shown. In this section we will only focus on the *System*-branch of the model. System is the tree's root feature, the child features are represented by the leaves of the tree, for example *Weight Opt.*, *Underlay* and *TC frequency*. The tree structure denotes the decomposition relation between the features. Child features can be either optional, marked by a blank circle, or mandatory, marked by a filled circle above. An optional feature is not required to be selected when the parent feature is selected, such as Weight Opt. in figure 2.2. Mandatory features on the other hand are required to be selected when the parent feature is selected, to make a valid configuration, such as Underlay and TC frequency in 2.2. [Pet90]

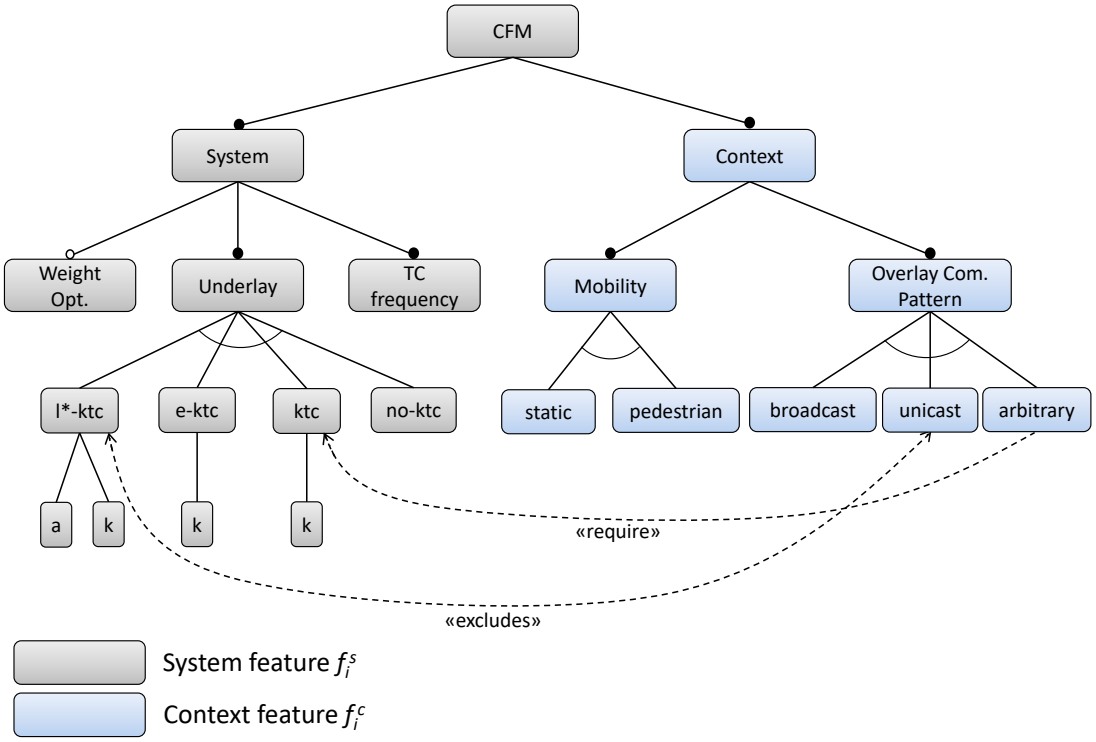

Figure 2.2: Example of a Context-Feature-Model using FODA notation.

Subsets of child features can be divided into or-groups and alternative groups. Or-groups are marked by a filled arc below the feature between all child feature links which are part of the or-group. An or-group requires at least one, and at most all child features in the group to be selected in a valid configuration if the parent feature is selected. Alternative groups are marked by an arc below the parent feature. The child features of Underlay in figure 2.2 form an alternative group. In an alternative group exactly one feature of said group has to be selected for a configuration. [Pet90]

Additionally cross-tree constraints are used to model relationships between features which are not contained in a tree decomposition hierarchy. These constraints include mutually exclusive statements and require statements. Two mutually exclusive features cannot be selected in a configuration at the same time. Mutually exclusive features are marked in the tree by a link with arrowheads at each end and *excludes* above the link. In figure 2.2 l*-ktc and unicast are mutually exclusive and can therefore not be selected in a valid configuration simultaneously. A feature can require another one to always be selected in a valid configuration, when the first feature is selected itself. Features which are required by other ones are marked by a link between the two, with an arrowhead pointing to the feature whose selection becomes required as well as the *require* tag above the link. The arbitrary feature in figure 2.2 requires the ktc feature to be selected as an example for a require cross-tree constraint.

These group and cross-tree constraints make model validation necessary to ensure the validity of any specific configuration, which is a specific selection of features. The constraints of a feature model confine the configuration space for a given SPL to contain only valid configurations. [Pet90] A feature model is used to specify the set of all valid configurations, which meet all the constraint requirements. The configurations can also be represented by means of algebraic representation or as an equivalent constraints problem. [WLS+16]

[ALHM+11] establishes a semantic mapping from feature model to propositional logic as is shown in figure 2.3. This mapping allows us to describe a given configuration space of a feature model as an equation using propositional logic. The propositional logic used includes implication ($\rightarrow$), bi-implication ($\leftrightarrow$), negation ($\neg$), conjunction ($\wedge$), (inclusive) disjunction ($\vee$) and exclusive disjunction ($\otimes$). An exclusive disjunction is hereby defined as a term which is only true if exactly one of the features, part of the term, is selected.

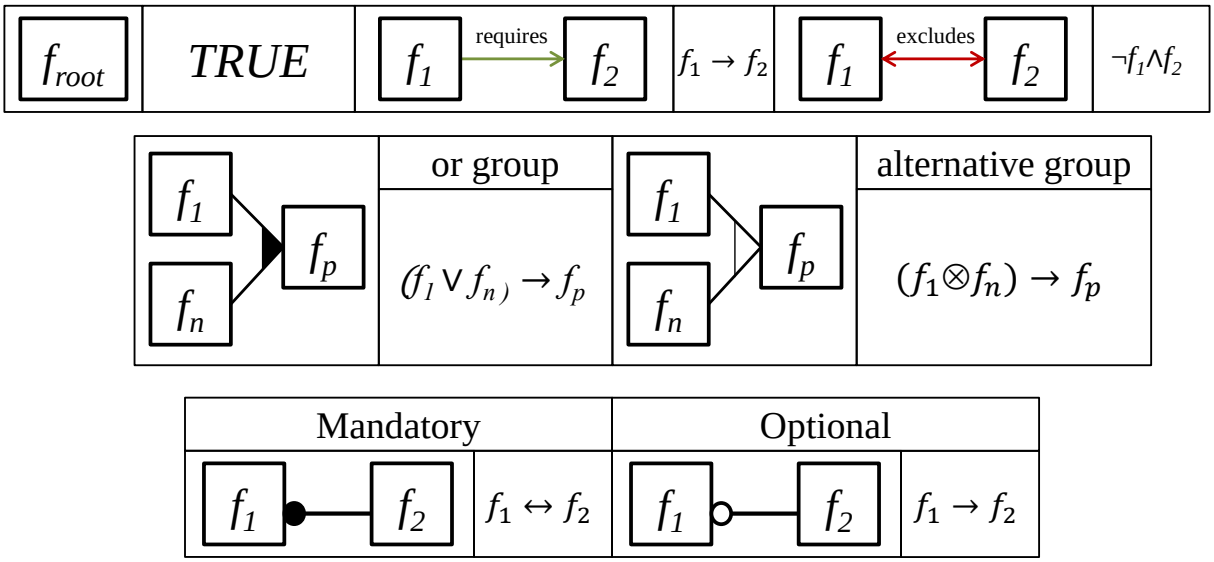

Figure 2.3: Mapping of a feature model to propositional logic [ALHM+11]

$$
\begin{aligned}
&1.(System \leftrightarrow true) \wedge \\
&2.(WeightOpt. \rightarrow System) \wedge (System \leftrightarrow (Underlay \wedge TCFrequency)) \wedge \\
&3.(Underlay \leftrightarrow (l^*\text{-}kTC \otimes e\text{-}kTC \otimes kTC \otimes non\text{-}kTC)) \wedge \\
&4.(l^*\text{-}kTC \leftrightarrow (a \wedge k)) \wedge \\
&5.(e\text{-}kTC \leftrightarrow k) \wedge \\
&6.(kTC \leftrightarrow k)
\end{aligned}
\tag{1}
$$

The first line of equation 1 defines that in all configurations the root feature *System* must be selected. The second line declares *WeightOpt.* as an optional feature, and *Underlay* and *TCFrequency* as mandatory features. The third line defines the alternative group of Underlay's child-features. The fourth, fifth, and sixth line define which features are mandatory for the respective Underlay child feature. All lines together describe the entire valid configuration space of the feature model. [ALHM+11]

2.2.3 Dynamic Software Product Lines

Modern software systems have higher requirements of adaptability at runtime. Conventional SPLs do not support a dynamic adaption of system configuration to meet changes after deployment. In order to address context-aware properties of the system, it is necessary for the system to evolve after deployment. To do so requires dynamic extensions, runtime reconfiguration, and autonomous system behavior to optimize the system's performance. For utilizing this dynamic functionality, the definition of context conditions or user preference options, defining the system's behavior, are necessary. Context is defined in [Pet90] as the circumstances, situation, or environment in which the given system exists. The feature model as used in SPL is expanded by this context, creating a context

feature model. With the given conditions and preferences the system can be adapted to reconfigure automatically and adjust to any given context optimally. [CBT+14]

DSPLs can for example enhance performance in a network of mobile devices. Depending on the location of the device and current state of the network, it may be beneficial to use WiFi Ad-Hoc communication instead of using only infrastructural ones. Due to the continuously changing context at runtime and the highly complex constraint solving problem, it is a challenge to optimally reconfigure at runtime without influencing the devices functionality and ensuring that all resource limitations are met. [SLR13]

2.2.4 Context Feature Models (CFM)

The goal of using a feature model is to select the combination of features in the model to meet the requirements given by an end user or customer. A Dynamic Software Product Line (DSPL) approach allows to not only specify a configuration during application engineering (SPL approach), but it also supports reconfiguration during runtime to allow for adaption. In a system with changing environmental influences, this allows for reconfiguration of the system to fit these changes by keeping the configuration valid. A context feature model (CFM) can be used to model a system and account for the changing environment, or context. A CFM is divided into two branches, a feature-model and a context-model. In figure 2.2 these two branches are specified by System and Context respectively. The context features are predefined but constantly changing due to changes in the system's environment and impose hard constraints on the system. [SLR13]

By specifying cross-tree constraints between these two branches such as shown in figure 2.2, and allowing run-time adaption, variable specification in regard to the environmental context can be modelled to match the requirements set by its current context. In figure 2.2 the overlay communications pattern arbitrary requires the underlay kTC to be selected for example. When the context changes to arbitrary in the context branch the system branch needs to be adapted during runtime to keep the configuration valid by selecting kTC in the system brand. However due to the immense amount of possible combinations and configurations, the task of finding optimal feature configurations for a certain context configuration is a challenge. As mentioned previously it is necessary to find constraints, which allow the system to make the choices of optimal feature selection dynamically without breaking resource limitations and keeping all other functionality working properly. [SLR13]

2.3 Non Functional Properties and Sampling Feature Models for Feature Interactions

2.3.1 Non-Functional Properties of SPLs

When commissioning a Product or choosing a specific variant of an SPL, stakeholders have certain requirements regarding non-functional properties (NFP). Some common NFPs include performance, reliability and footprint. Often these requirements can be conflicting. The kTC topology control algorithm as mentioned in section 2.1 could for example be beneficial to select when energy efficiency is important, since it removes long links which require high transmission power. When removing links however the reliability of the

system is reduced, since a dropped link could cause some nodes to be disconnected. Maxpower topology control on the other hand would ensure high reliability, because the algorithm activates all available links in the topology thus allowing the network to keep functioning properly, even if a link is dropped [KSV+17]. In figure 2.4 two different topology control mechanisms can be seen. In Variant A, which resembles maxpower topology control, there are a lot of redundant links present in the topology resulting in a strong reliability. Variant B has less links and little redundancy. Other NFPs such as optimizing memory consumption by selecting certain features could for example impair footprint optimization. A trade off between different NFPs is therefore often necessary. Different stakeholders can also have interests in optimizing different NFPs, since their requirements may vary. [SRK+11]

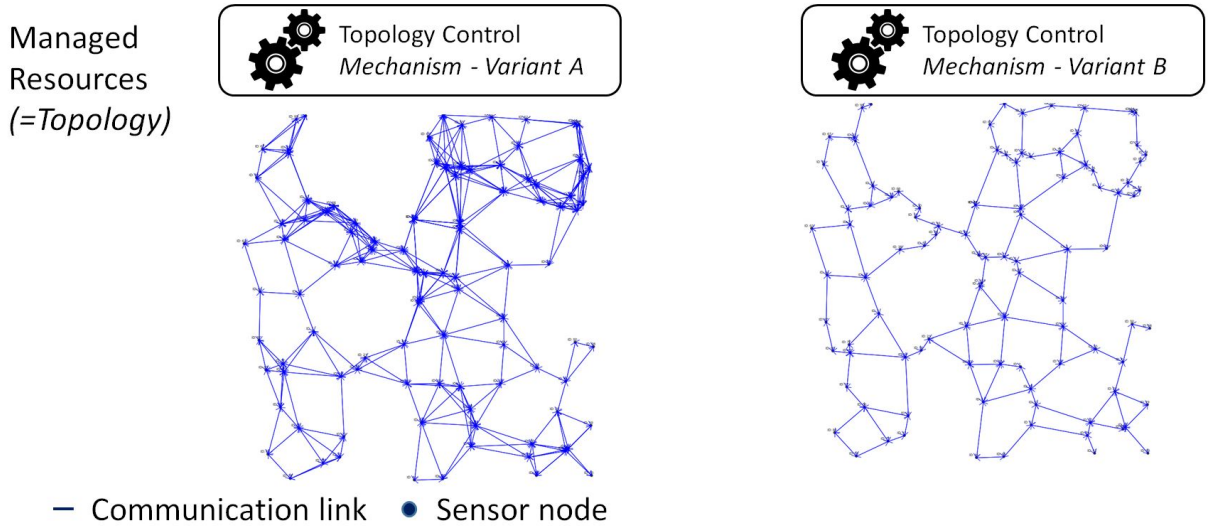

Figure 2.4: Two different topology control mechanisms and their effect on the network topology

The exact NFPs are not always known until after development, and therefore an optimization during the development phase is not possible. In most SPLs it is impossible to generate, compile and execute each variant of the valid variant space due to the enormous number of variants. SPL Conqueror proposes a holistic approach to optimize an SPL in regard to NFPs [SRK+11]. These properties are divided into three categories: qualitative, feature-wise quantifiable and variant-wise quantifiable properties. Qualitative properties are those that can not be measured by a quantitative metric. These properties have to be ranked on an ordinal scale which usually requires domain knowledge. Examples of qualitative properties include reliability, security and usability. Feature-wise quantifiable properties can be measured on a metric scale. Some examples are footprint and accuracy. After measurement each feature receives a specific value depending on its influence on the desired property. An aggregation function is defined to aggregate the values of all selected features in a specific variant and compare different variants. Lastly variant-wise quantifiable properties are properties with no meaning for a single feature but rather for the entire set of features selected in a specific variant. To measure these properties each

variant has to be generated. Since it is inefficient and expensive to generate every single variant, only a predefined set of variants is used for measurement. The measurement is often done by benchmarks and performance is the most common property in this category. [SRK+11]

2.3.2 Feature Interaction

In a given software system the NFPs can be influenced by selecting features which interact. Feature interaction describes features which behave differently when other features are present than they would on their own. In [SKK+12] an example of feature interaction is given by a flood-control sensor (f_c) working together with a fire-alarm sensor (f_a). If only one of the sensors is present they will simply turn the water off when flooding is detected or on when a fire is detected respectively. When both sensors are working together however there is an interaction between f_c and f_a which turns off water after a fire has been detected, if flooding is detected. This interaction is written as $f_c \# f_a$. Features can be pairwise dependent as shown in the example or t-wise dependent, $f_x \# f_y \# f_z$. Not all feature interactions have an observable effect on NFPs. It is therefore relevant to find only the ones with significant effect. [SKK+12] uses the following notation for a program P consisting of features f_c and f_a without feature interaction: $\mathrm{P} = f_c \cdot f_a$. When interaction is taken into account the term $f_c \times f_a = f_c \# f_a \cdot f_c \cdot f_a$ describes a given configuration with all features and feature interactions. [SKK+12]

SPL Conqueror supports all three of the characterized property categories to find the optimal variant. The user has to define the measurement procedure for the SPL and SPL Conqueror performs the process of selecting, generating and measuring features automatically. Feature interactions alter an SPL's non functional properties depending on the presence of a specific feature combination. SPL Conqueror models this interaction and utilizes it to predict the behavior of NFPs of different variants. Features sharing common code can be a positive effect of feature interaction. Negative effects include the requirement of additional code or bus overload for certain feature combinations. [SRK+11]

2.3.3 Sampling

SPL Conqueror employs stepwise linear regression on a sample set of measured configurations of a performance-influence model in order to learn its function. For sampling of the model it has the options of using random, binary and numeric sampling. For random sampling configurations are chosen at random. This sampling method has its limitations since choosing valid configurations becomes difficult with more constraints present. For some solvers random sampling leads to valid configurations of a small, clustered solution space in the entire valid configuration space. Binary sampling uses a heuristic which picks valid configurations and then learns the basic influence of every individual feature by selecting and deselecting it. Featurewise sampling selects configurations in a way to avoid interactions, by finding ones with as many features deselected as possible. This process is repeated until all configurations have been included. Negative featurewise sampling tries to maximize the amount of interaction in the configurations by picking the ones with as many features selected as possible. The last binary sampling method is pairwise sampling.

Here only configurations with two-way interactions are taken into consideration, to not be influenced by other interactions. [SGAK15]

Numeric sampling utilizes an approach known as Design of Experiment in order to generate an experimental plan assigning values to configurations. One example used by SPL Conqueror is the Plackett-Burman Design. The design uses a limited number of measurements to minimize the variance of the independent variables. It defines seeds, which are dependent on the number of experiments conducted, as well as levels used for distinctive values of an independent variable. These values are used to determine if feature interactions are present based on domain knowledge of their presence. [SGAK15]

3 Mathematical Model for Optimal Identification of Configurations

The goal of this thesis is to take large amounts of measurements, either in the form of simulated or actual data, and encode the aggregated information as a mathematical optimization model, to find the optimal configuration of a DSPL in regards to a measured NFP. We assume that interactions between system and context features have an influence on certain NFPs. We therefore want to find the system feature selection which interacts optimally with the selected context features for a given NFP, while remaining within the valid configuration space. The mathematical optimization model includes constraints confining the configuration space ,and an objective function used for finding the optimal selection of features. To develop and evaluate the optimization model, simulation data is generated.

3.1 Defining the Configuration Space using FMs

To define the configuration space for a given system we will be using FMs. Figure 3.1 shows the FM used for the first set of simulation data (D4wsntraces). In order to define FMs semantically we will be using the CardyGAn Tool Suite. The CardyGAn Tool Suite provides the CardyGAn-Model interface, for defining FMs using cardinalities, and CardyGAn-ILP which, provides an interace for translating propositional logic into ILP and solving it using an objective function.

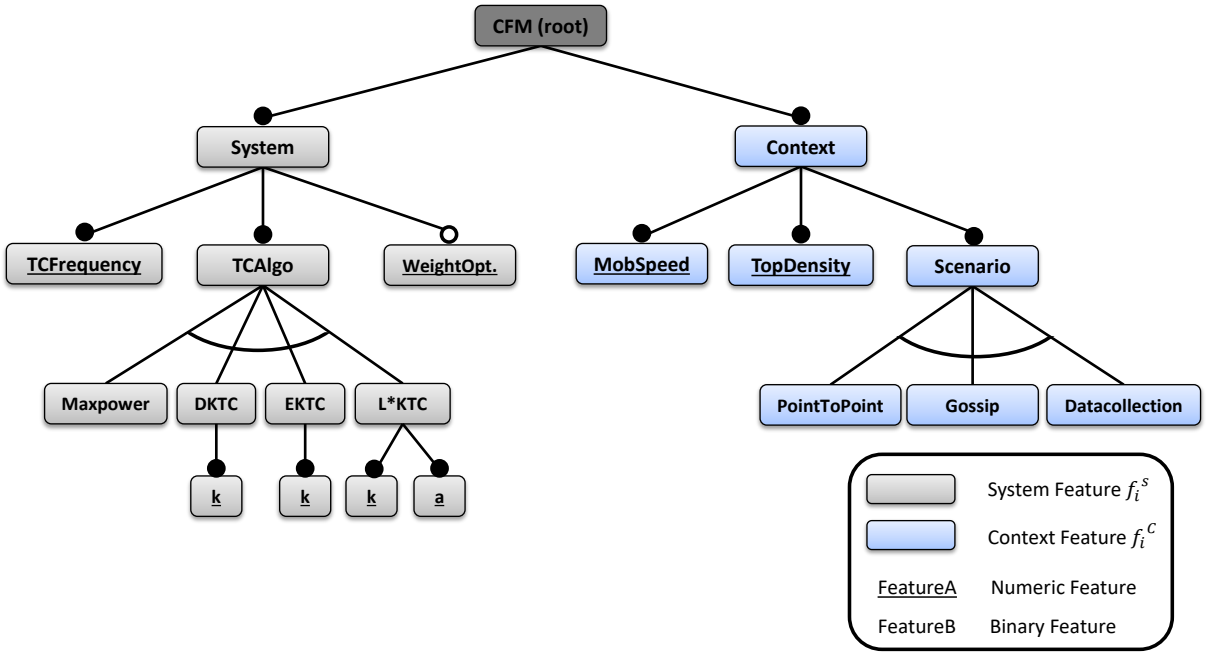

Figure 3.1: The FM used for the D4wsntraces simulation data.

3.1.1 Using a Cardinality-Based Approach for defining FMs

The FMs are first defined using cardinality-based definitions, as provided by CardyGAn-Model. Cardinality-based FMs contain intervals defining the different constraints making

up an FM. They can be used to describe FMs containing multiple instances of the same feature. For this thesis however, we will only be considering FMs containing at most one instance of each feature. Figure 3.2 shows a minimal example of a cardinality-based FM. It is based on the constructs provided in [WLS+16].

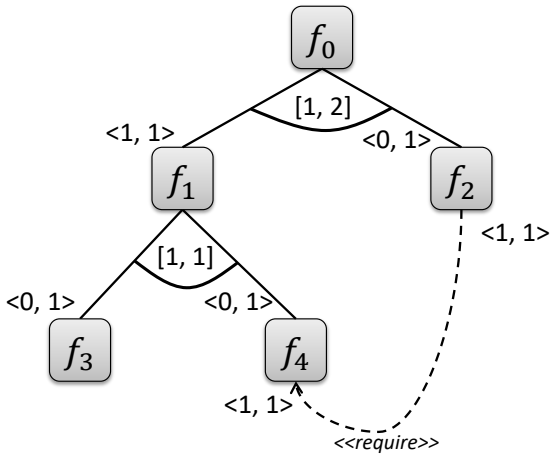

Figure 3.2: A simple example of an FM using cardinality.

For the FMs used in this thesis the relevant constructs are *feature instance cardinality*, *feature group type cardinality* and *cross-tree edges*.

- The feature instance cardinality, marked as $< l, u >$ on the top left of each feature, restricts the minimum and maximum number of feature instance which can be selected. For our models the interval $< 0, 1 >$ describes an optional feature and the interval $< 1, 1 >$ describes a mandatory feature. In figure 3.2, an example of an optional feature is f_2 and an example of a mandatory feature is f_1.

- The feature group type cardinality, marked as $[l, u]$, restricts the number of types of sub-features from a selected feature which can be selected. In figure 3.2 the feature group type cardinailty of f_1 is $[1, 1]$, representing an alternative group (XOR), where exactly one of the sub features f_3 or f_4 has to be selected. The feature group type cardinality of f_0 ($[1, 2]$) allows either one or two sub-features to be selected. The feature instance cardinality of f_1 restricts it to be selected in all valid configurations, therefore the two options for the selection of sub-features of f_0 are: f_1 is selected and f_2 is not selected or f_1 and f_2 are selected.

- Cross-tree edges, consisting of *require* and `exclude` edges, are marked by a dotted line between the target and source feature. The $< l, u >$ constraints are marked at both ends of the connecting line and define constraints on the number of instances of arbitrary pairs of features. For the models used in this thesis all cross-tree edges are marked by $< 1, 1 >$ since we only allow one instance of each feature to be selected at any time. In figure 3.2 f_2 and f_4 are connected by a require cross-tree edge. If the optional feature f_2 is selected in a configuration, then f_4 must also be selected as part of this configuration for it to be valid.

3.1.2 FM definition by Propositional Logic

We want to optimize a given NFP by picking the optimal configuration of features within the valid configuration space. To achieve this CardyGAn-ILP provides both a framework for defining FMs as propositional logic, and a translation of the defined propositional logic into ILP. The propositional logic defines the constraints which restrict the solution space of the ILP to the valid configuration space of the given FM. CardyGAn-ILP uses an objective function, specifying a certain goal, to find the optimal solution from the solution space.

The rules used for defining an FM as propositional logic are given in section 2.2.2 and the included figure 2.3. Equation 2 shows the propositional logic representation of the FM in figure 3.1. This definition provides the first part of the optimization model, defining the valid configuration space of the model used.

$$
\begin{aligned}
&(root \leftrightarrow true) \wedge \\
&(System \leftrightarrow root) \wedge \\
&(TCFrequency \leftrightarrow System) \wedge (TCAlgo \leftrightarrow System) \wedge (WeightOpt. \rightarrow System) \wedge \\
&(Maxpower \otimes DKTC \otimes EKTC \otimes L^*KTC \rightarrow TCAlgo) \wedge \\
&(DKTC \leftrightarrow k) \wedge \\
&(EKTC \leftrightarrow k) \wedge \\
&(L^*KTC \leftrightarrow k) \wedge (L^*KTC \leftrightarrow a) \wedge \\
&(Context \leftrightarrow root) \wedge \\
&(MobSpeed \leftrightarrow Context) \wedge (TopDensity \leftrightarrow Context) \wedge (Scenario \leftrightarrow Context) \wedge \\
&(PointToPoint \otimes Gossip \otimes Datacollection \rightarrow Scenario)
\end{aligned}
\tag{2}
$$

The constraints confining the model can also include cross-tree constraints. Besides the model used as part of the simulation data, a second similar model is designed to create synthetic data, for precise evaluation of the optimization at the end. The FM used for this synthetic data is shown in figure 3.3 and contains both require and exclude cross-tree constraints. The FM is slightly adjusted from the FM used to create the D4wsntraces simulation data. Exclude constraints are marked red in the figure with an arrow head at both ends of the link. Require constraints are marked green with an arrow pointing to the required feature. A model for synthetic data creating provides the advantage of being able to manually set feature interactions which can be used in the evaluation of the correctness of the optimization provided by this model.

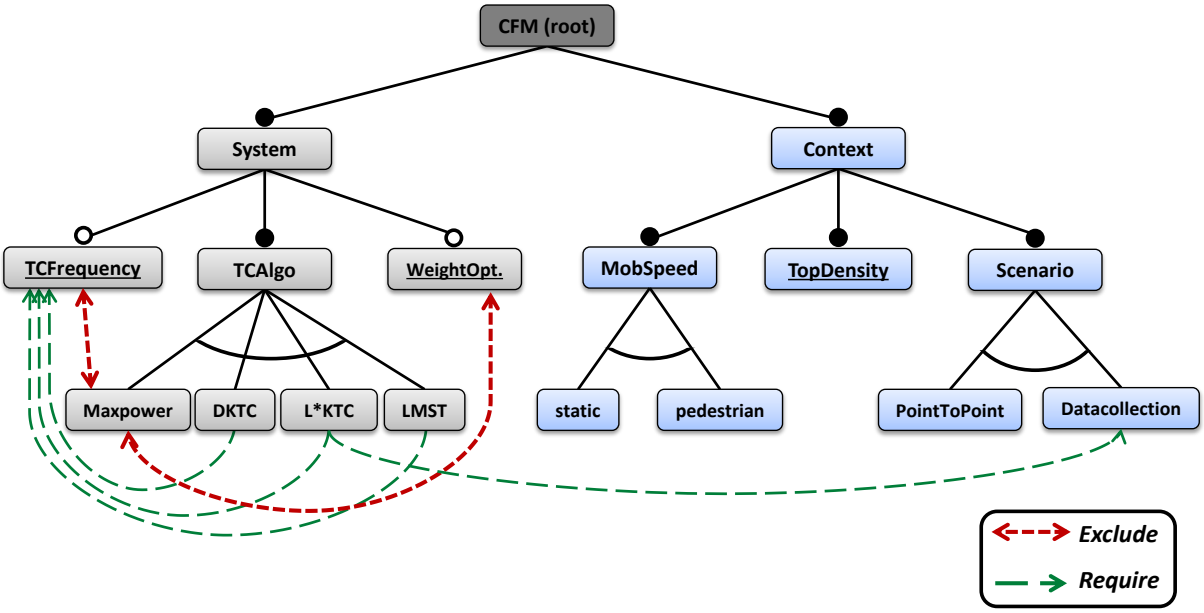

Figure 3.3: The FM (titled D4x) used to create synthetic data containing manually set feature interactions.

The propositional logic representation of the cross-tree constraints is shown in equation 3. The first line defines the two excluded edges using the previously defined propositional logic for cross-tree constraints. The other lines define the multiple require edges contained in the model. The entire propositional logic representation of an FM consists of both the regular constraints and the cross-tree constraints.

$$\begin{aligned}&(\neg Maxpower \wedge TCFrequency) \wedge (\neg Maxpower \wedge WeightOpt.) \wedge \\ &(DKTC \rightarrow TCFrequency) \wedge \\ &(L^*KTC \rightarrow TCFrequency) \wedge (L^*KTC \rightarrow Datacollection) \wedge \\ &(LMST \rightarrow TCFrequency)\end{aligned} \tag{3}$$

3.2 Defining the Objective Function using Feature Interactions

The context of a given model is always dependant on the current environment and can not be set by the user. The part of the model the user can selectively configure are the system features. We want to find all interactions between the user configurable system features and the environmentally given context features which have a positive or negative influence on a certain NFP, in order to use them in the optimization model. An optimized model will consist of hard constraints given by the model definition (FM) and the soft constraints obtained from the optimization model. The hard constraints ensure the selection of a valid configuration and the soft constraints lead to the optimal configuration within this selection space in regards to the NFP used for finding them and the current context.

For the objective function we propose a weight function connecting the feature interactions and their relative influence on the given NFP. For each interaction an additional

decision variable (t_i) is introduced representing an indicator, which shows whether all features part of that interaction are selected. If all features part of the interaction are selected, the new decision variable is selected, otherwise it is not. For an interaction between system feature f_1 and context feature c_1, the definition of the decision variable is shown in equation 4 as an example using propositional logic. As a result of this translation from a feature interaction to a decision variable, the set of feature interactions present corresponds to a set of decision variables representing those feature interactions.

$$f_1 \wedge c_1 \rightarrow t_1 = 1 \text{ (otherwise } t_1 = 0) \tag{4}$$

In the optimization model, the relative influence of the interactions between system features and context features needs to be defined. Each interaction is assigned a weight (ω_i) corresponding to its quantified influence on the NFP. The newly introduced decision variables and weights are used by the optimization model in its objective function as the proposed weight function. The objective function, as defined in equation 5, is the sum of each weight multiplied by the decision variable of the corresponding interaction. The optimization model returns the optimal selection of decision variables according to the objective function, the constraints given through the model definition and the choice of maximization or minimization. The resulting decision variables represent the optimal selection of soft constraints for optimizing the model in regard to the given NFP.

$$ObjectiveFunction = min/max \sum \omega_i t_i \tag{5}$$

3.3 Overview of the entire Optimization Model

The entire optimization process consists of defining the configuration constraints laid out by the given FM, finding interactions between system features and context features and incorporating these interactions into an objective function, which is solved using integer linear programming (ILP). We will be using SPL Conqueror to find the value of the weights used for representing the impact of an interaction on an NFP. SPL Conqueror is a library for learning the influence of different configurations on NFPs. It creates an influence model, which contains terms representing the relative influence of each feature or interaction. An iterative learning process using linear regression and step-wise feature selection is used for the learning process [Sie]. We use SPL Conqueror for finding the feature interactions present in the measurements made with our simulation data. The terms contained in the influence model are used as the weights of the objective function. The integration of SPL Conqueror's results will be discussed in the next chapter.

A flowchart of the proposed approach for creating the optimization model is shown in figure 4.1. For defining the constraints of the FM we will be using the CardyGAn Tool Suite [SWK+16]. It contains a cardinality-based component and a component using propositional logic for conversion into ILP. The measurement data used in the optimization is contained in a comma seperated value (CSV) file. For finding the feature interactions and measuring their impact on an NFP SPL Conqueror is used. The following sections will describe each step of the process shown in the flowchart in detail. In the flowchart each step implemented using Java is marked by a red arrow. The step implemented using Python is marked by a green arrow. Boxes colored in blue represent Java objects of the Class given in the name and boxes colored in green represent files such as xml and csv. The numbers marking the links correspond to the section of this chapter which discusses the implementation of the given link.

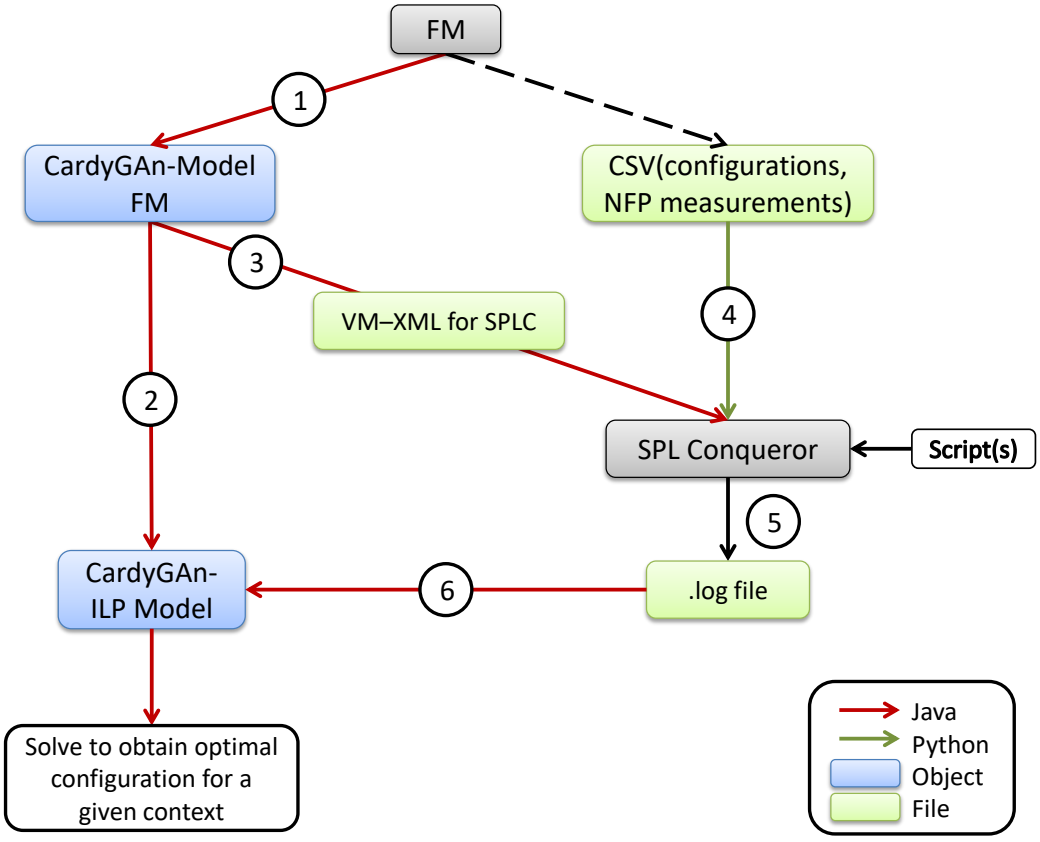

Figure 4.1: Flowchart showing the implementation of the entire process of making the optimization model.

The CardyGAn-Model interface provides a factory-pattern for creating an FM object and adding feature objects and cross-tree edge objects to it. Figure 4.2 shows an excerpt of the UML diagram showing the factory pattern used in CardyGAn-Model. In order to create an FM using CardyGAn-Model, first an *FmFactory* object must be instantiated. The *FmFactory* interface contains methods for creating *FM*, *feature* and *CrossTreeConstraint* objects. An *FM* object contains the root feature and the cross-tree constraints. The *Feature* objects contain the feature's parent, the list of child features and the different cardinalities, feature instance and group type.

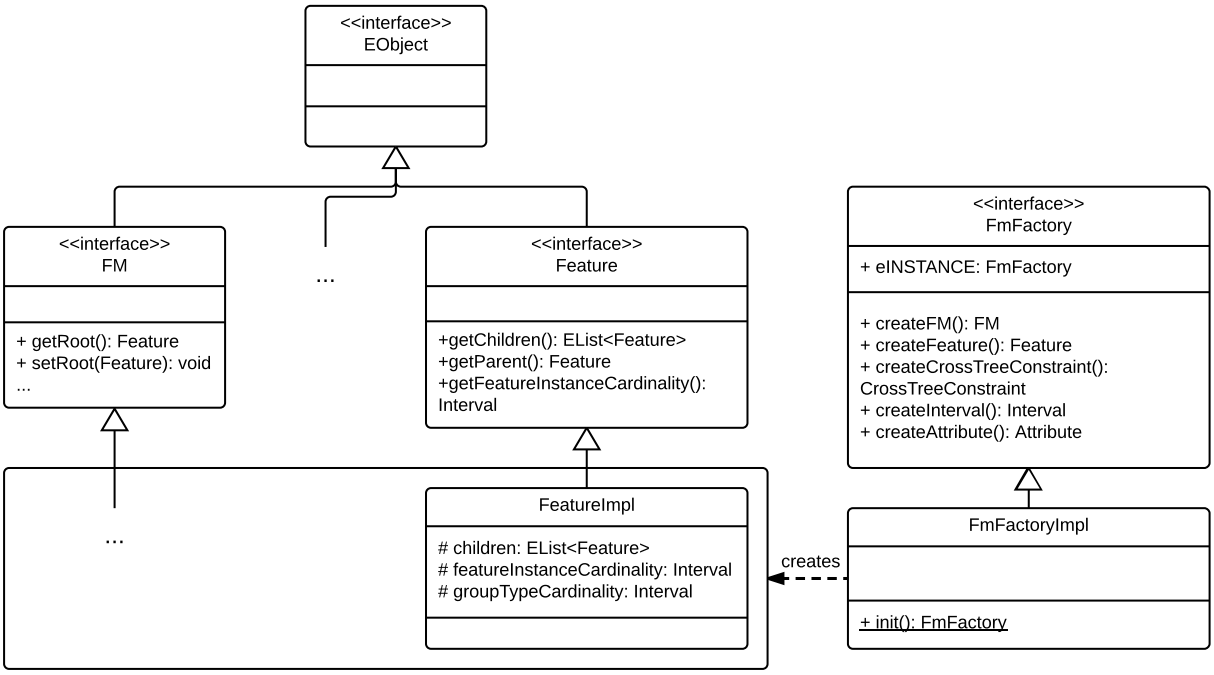

Figure 4.2: UML diagram showing an excerpt of the factory pattern used in CardyGAn-Model.

Listing 4.1 shows a code excerpt from the implementation of an FM in CardyGAn-Model. In line the *FmFactory* object is created. In the following lines an *FM* and a *Feature* object are created using the factory. In line six the name of the feature *root* is set and in the seventh line the *Feature* root is set as the root feature of the *FM* object fm. The last line uses the newly defined methode *makeInterval(int lower, int upper)* to create an *Interval* object with lower bounds set to the int value *lower* and upper bounds set to the int value *upper*. The feature instance cardinality is then set to that *Interval*, in this case [1, 1].

Listing 4.1: Java code excerpt implementing an FM using CardyGAn-Model

```
1 FmFactory myFactory = FmFactory.eINSTANCE;
2
3 FM fm = factory.createFM();
4
5 Feature root = factory.createFeature();
6 root.setName("root");
7 fm.setRoot(root);
8 root.setFeatureInstanceCardinality(makeInterval(1, 1));
```

For developing and later evaluating the implementation of the optimization model, different sets of simulation data were used. These sets of data were each created using an FM which was defined using the FODA notation, introduced in chapter 2. In order to define these models using the cardinality-based approach of the CardyGAn-Model framework a set of rules for converting the FODA notation to the cardinality-based definition is required. We want to be able to define each model once using CardyGAn-Model and then translate it into the other necessary formats for a faster and easier work flow.

In the models used for the simulation data all features are either optional or mandatory. Figure 4.3 shows the conversion of optional and mandatory features from the FODA notation to the cardinality-based one. In figure 4.4 the conversion of an alternative group into cardinality-based notation is shown. For an alternative group the feature group type cardinality of the feature is set to the interval $[1, 1]$ as shown in the figure. The figure also shows the implementation of numeric features in the framework. A numeric feature contains upper and lower bounds ($[l, u]$) representing the numeric range of the feature. The bounds are held by the numeric feature as part of an attribute which is represented by a red oval in the figure.

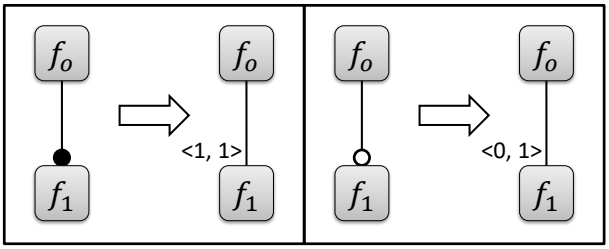

Figure 4.3: Conversion between optional and mandatory features from FODA to cardinality-based notation.

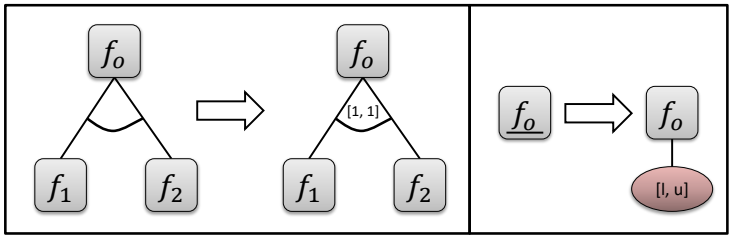

Figure 4.4: Coversion of an alternative group from FODA notation to cardinality-based notation and implementation of a numeric feature in the CardyGAn-Model framework.

While feature instance group cardinaltiy and feature group type cardinality are both stored as part of each feature, the cross-tree edges are created as seperate objects. A *CrossTreeConstraint* object contains the type of the constraint, either *REQUIRE* or *EXCLUDE*, as well as the source and the target of the constraint. For require constraints the target is the feature to which the connecting arrow is pointing and the source is the other feature. For exclude constraints source and target are interchangeable, their order has no significance.

Using these conversion rules the FM for the first set of simulation data (D4wsntraces), as shown in figure 3.1, can be converted into a cardinality-based model. The model is also expanded to contain the attributes used for defining the numeric features' value ranges. Figure 4.5 shows the entire FM graphically as it is defined using CardyGAn-Model. The lower and upper bounds of feature group type cardinalities from features with only mandatory sub-features are always both equal to the number of sub-features. Both *System* and *Context* are mandatory features, as marked by the $< 1, 1 >$ feature instance cardinality, thus the feature group type cardinality of the root feature is $[2, 2]$. *System* has two mandatory and one optional sub-feature, thus the lower bound is equal to the amount of mandatory sub-features and the upper bound equals the total amount of sub-features, resulting in a feature group type cardinality of $[2, 3]$. The maximum and minimum bounds contained in the numeric features' attributes are obtained from the simulation data.

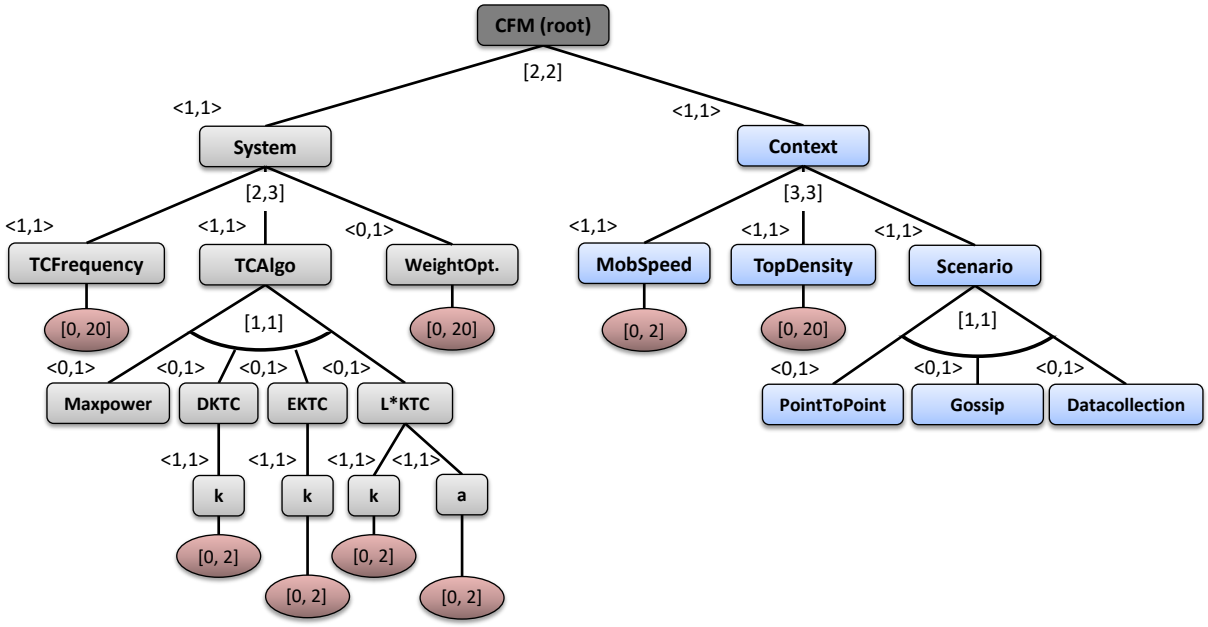

Figure 4.5: Graphical representation of the FM used for the D4wsntraces simulation data as it is defined using CardyGAn-Model.

4.2 Converting from Cardinality-Based models into Propositional Logic

We introduced propositional logic for defining FMs as part of the final optimization model in the previous chapter. The optimization model uses propositional logic for defining the constraints of the FM used to restrict the solution space to the valid configuration space. For defining the propositional logic we will be using CardyGAn-ILP, which is part of the previously introduced CardyGAn Tool Suite. CardyGAn-ILP provides a translation of propositional logic to integer linear programming (ILP).

The first step of using CardyGAn-ILP is to define the FM as propositional logic. We want to manually define each FM only once using CardyGAn-Model, as discussed in the previous section. To obtain the model in the form of propositional logic, we will therefore devise a set of translation rules. This enables us to use the cardinality-based definition, previously established, as the input, and output the model as defined by propositional logic. This step of the complete optimization process is labeled 2 in figure 4.1, connecting CardyGAn-Model and CardyGAn-ILP.

The basic translation rules for converting the feature instance cardinality are shown in figure 4.6. All features used as part of the models in this thesis are either mandatory or optional, labeled as $< 1, 1 >$ and $< 0, 1 >$ respectively. In propositional logic mandatory features are translated into a bi-implication, meaning that if f_0 is selected, then f_1 must also be selected and if f_1 is selected, then f_0 must also be selected. As a result neither feature can be selected while the other is not. Optional features are translated into an implication from optional feature to its parent feature. Consequently, if f_0 is selected, f_1 can be either selected or deselected, but if f_1 is selected f_0 must also be selected.

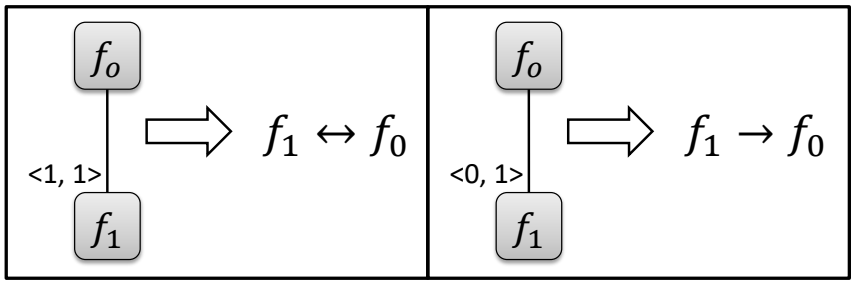

Figure 4.6: Basic translation rules from a cardinality-based model to propositional logic, for mandatory and optional features.

In figure 4.7 the translation of an alternative group (XOR) is shown. As previously defined in the cardinality-based model, an alternative group is represented by the feature group type cardinality interval $[1, 1]$. In propositional logic the alternative group is expressed by a bi-implication between the feature and an exclusive disjunction of the sub-features. Since all models used as part of this thesis only contain alternative groups, the translation of other feature group type cardinalities were simply done using the basic translation rules of madatory and optional features.

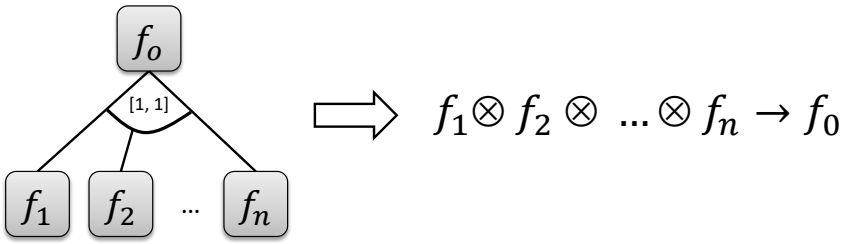

Figure 4.7: Translation of an alternative group into propositional logic.

For defining the features of the model CardyGAn-ILP provides two variable types, binary variables (*BinaryVar*) and integer variables (*IntVar*). Binary features are translated into BinaryVars and numeric features are translated into IntVars. To obtain objects of these variables a *Model* object is instantiated which contains methods for creating them, as well as constraint objects for specifying the propositional logic, which defines the model. For the alternative group shown in figure 4.7 the expression used for defining the constraint is given in listing 4.2. The first term of the bi-implication is the parent feature. The second term restricts the sum of the sub-features to one, creating an exclusive disjunction.

Listing 4.2: Expression used to specify an alternative group in CardyGAn-ILP

```
1 bi_impl(f0, eq(sum(f1, f2, ..., fn), param(1))
```

To complete the translation from cardinality-based models to propositional logic, cross-tree constraints must be translated. Require cross tree constraints are translated into an implication from source to target, as shown in the top of figure 4.8. The bottom of the figure shows the translation of an exclude constraint into a negation of the conjunction of source and target feature.

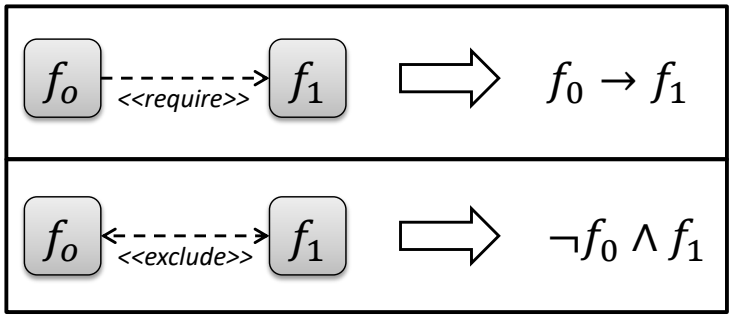

Figure 4.8: Translation of cross tree constraints into propositional logic.

As stated before CardyGAn-ILP provides a translation of the given propositional logic into ILP. CardyGAn-ILP uses an objective function to define the optimization goal. Using the propositional logic for restricting the solution space of the optimization to the valid configuration space, CardyGAn-ILP uses the objective function to find the optimal solution from this solution space. Figure 4.9 shows the translation of a simple cardinality-based model into propositional logic, using the previously defined rules. An example of an objective function would be a function maximizing the number of selected features. For this example the optimal solution would be the configuration in which f_0, f_1, f_2 and f_4 are selected. Since f_2 requires f_4 to be selected, any configuration in which f_3 is selected can not have f_2 selected, thereby having less than the maximum possible number of features selected.

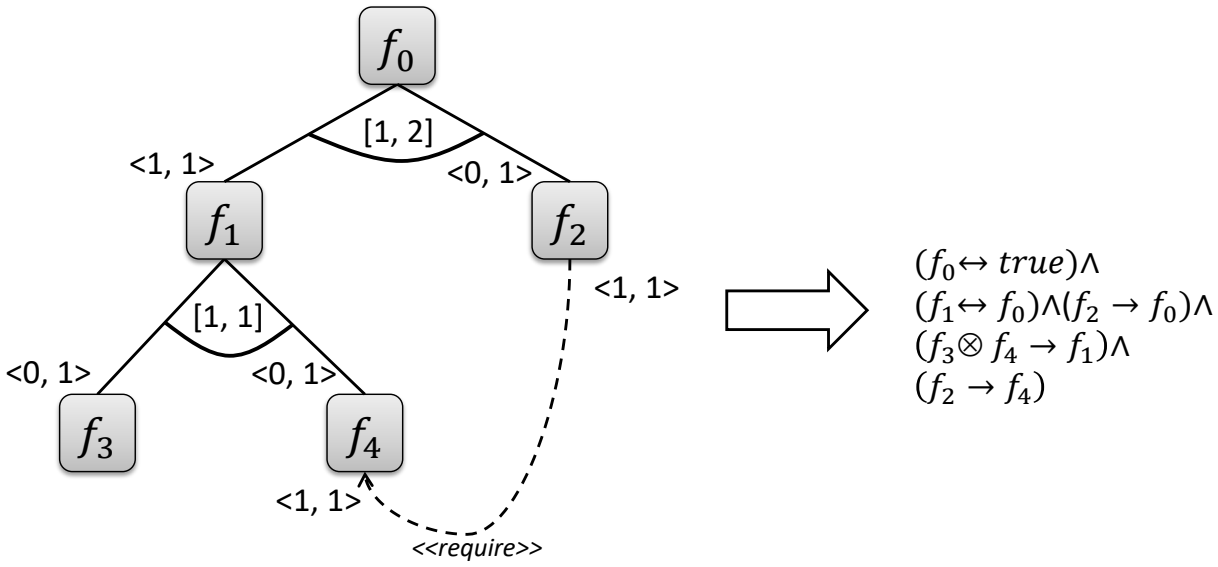

Figure 4.9: Translation of the simple cardinality-based model used in section 4.1 into propositional logic.

4.3 Generating an XML Representation of FMs for SPL Conqueror

SPL Conqueror requires a definition of the FM used to make the NFP measurements for the creation of its influence model using Extensible Markup Language (XML). SPL Conqueror provides a GUI for creating the XML file describing an FM. The GUI can be used for creating a model and adding features and constraints to it. The definition of the FM is already contained in the definition made with CardyGAn-Model. Therefore a translation from the cardinality-based CardyGAn-Model definition to a definition in the XML format is developed. By simply translating the already existing definition, the GUI provided by SPL Conqueror does not need to be used. This section will discuss this translation process, which is labeled by number 3 in the flowchart shown in figure 4.1.

Listing 4.3 gives an excerpt of the XML file which describes the FM shown on the left in figure 4.9. SPL Conqeuror uses the term *variability model* to describe FMs. Each XML file describing the variability model starts with a *vm* tag containing the attribute *name*, which sets the models name. The *vm* tag contains the elements *binaryOptions* and *numericOptions*. The other elements are not relevant for the models used in this thesis.

Listing 4.3: Excerpt of the XML definition for the FM given in 4.9

```
<vm name="sampleVM">
    <binaryOptions>
        <configurationOption>
            <name>f0</name>
            <outputString/>
            <prefix/>
            <postfix/>
            <parent/>
            <children>
                <option>f1</option>
                <option>f2</option>
            </children>
            <impliedOptions/>
            <excludedOptions/>
            <defaultValue>Selected</defaultValue>
            <optional>False</optional>
        </configurationOption>
        .
        .
        .
    </binaryOptions>
    <numericOptions/>
    <booleanConstraints/>
    <nonBooleanConstraints/>
</vm>
```

BinaryOptions describes the binary features of the model. Each feature is defined using a *configurationOption* element containing child elements for specifying the different

characteristics of the feature. The child elements relevant for the translation are explained below:

- name: The name of the feature.
- parent: The name of the parent feature. For f_1 this element would be <parent>f0</parent>
- children: The name of each child feature is given in a child element *option* (lines 10 and 11).
- impliedOptions: Features which are required to be selected for a valid configuration if the feature being defined is selected.
  For f_2 impliedOptions would contain the child element <option>f4</option>.
- excludedOptions: Features which can not be selected in the same valid configuration as the feature being defined.
- optional: *True* if the feature is optional (feature instance cardinality $< 0, 1 >$), else *False* (mandatory feature $< 1, 1 >$).

For translating the cardinality-based model into the XML format a set containing all features is created by traversing over the tree which makes up the FM. For each feature from this set a new *configurationOption* element is created. The information about the feature's name, its parent, and its children is stored in the feature object created using CardyGAn-Model. For the implied options, the list of cross-tree constraints, which is part of the model, is traversed to find all constraints of the type *require* with the current feature as their source. The target feature for each of the constraints found is included in the *impliedOptions*. The list of cross-tree constraints is also traversed to find all constraints of type *exclude* which have the current feature as either their source or target. For each constraint meeting these requirements the other feature included in the constraint is added to the excludedOptions. To determine if the feature is optional the feature instance cardinality is used. If the interval is $< 0, 1 >$ the element *optional* is set to *True*, otherwise it is set to *False*. If the feature group type cardinality of a feature's parent feature is $< 1, 1 >$ (alternative group), then the feature must include all of the parent's other child features as part of the excludedOptions.

NumericOptions describes the numeric features of the model. For numericOptions the configurationOptions do not contain the child elements defaultValue and optional, but are expanded by the following child elements:

- minValue: The minimum value in the value range of the numeric feature.
- maxValue: The maximum value in the value range of the numeric feature.
- stepFunction: The function used for calculating the values used for the learning process. For a numeric feature n0, <stepFunction>n0 + 1</stepFunction>, starts at the minValue and adds 1 until the maxValue is reached. This example iterates over all integer values between the minimum and maximum.

For the numeric features the translation is done mostly the same as for binary features, except for the elements which are removed and the specific numeric elements. Using the the attributes included as part of the CardyGAn-Model definition, the minValue and maxValue can be set using the lower bounds and upper bounds values respectively. The stepFunction will always be set to the name of the feature +1, to iterate over all integer values within the range.

One additional rule is required for optional numeric features. An example of this would be *WeightOpt* from the FM used as part of the D4wsntraces simulation data. SPL Conqueror does not support the definition of optional numeric features. Therefore the feature needs to be redefined as shown in figure 4.10. An additional optional binary feature, *WeightOptBinary*, is introduced. The old feature *WeightOpt* is added as a numeric mandatory child feature of the newly created optional binary feature. SPL Conqueror supports all of the newly created features and the resulting construct used achieves the same results as the original would.

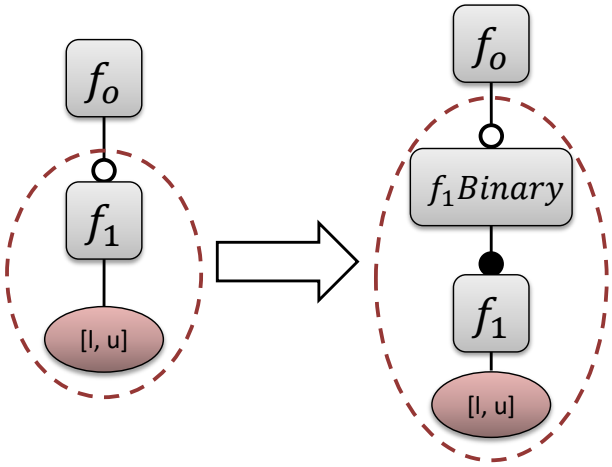

Figure 4.10: Excerpt of an FM showing adjustment of optional numeric features

The simulation data created for developing and evaluating the optimization process, labeled CSV in the flowchart shown in figure 4.1, is provided as comma separated value (CSV) files. Figure 4.11 shows a an excerpt of the CSV containing the simulation data. In the complete data set, each selected configuration is measured over five different seeds, which change the initial topology composition at the start of the measurement. The complete data set therefore includes five rows for each row shown in the excerpt, one for each of the five seeds. The naming scheme in the header of the file is used to differentiate between system features (fs), context features (fc), and metrics (m). The metrics represent the NFPs and each metric column contains the measurements made in the simulation.

Row	fsTCAlgo	fsTCParamK	...	fcMobSpeed	fcScenario	...	mEmean	...
1	MAXPOWER	NaN		0.0	POINTTOPOINT		99715493.46	
2	MAXPOWER	NaN		1.5	POINTTOPOINT		99713809.10	
3	DKTC	1.00		0.0	POINTTOPOINT		99715493.46	
4	DKTC	1.00		1.5	POINTTOPOINT		99713809.10	
5	DKTC	2.00		0.0	POINTTOPOINT		97800715.74	
6	DKTC	2.00		1.5	POINTTOPOINT		99713809.10	
7	EKTC	1.00		0.0	POINTTOPOINT		99715493.46	
...								

Figure 4.11: Table showing an excerpt of the D4wsntraces simulation data.

SPL Conqueror is able to use data stored as a CSV file for creating its influence model, however it needs to be in a specific format. Each feature contained in the variability model defined in the previous section needs to be included as a column, containing either if it is selected (1) or not (0), or the numeric value used in that configuration for numeric features. It is also important to note, that the name of the feature in the column header and the name used in the definition of the variability model must match. The columns containing the NFP measurements are included after the feature columns. In the table shown in figure 4.11 features which are part of an alternative group in the FM are given using their name as the value in the column representing their parent feature. In the table the system feature *fsTCAlgo* and the context feature *fcScenario* represent alternative groups. In order for SPL Conqueror to understand the configurations correctly, the CSV needs to be adjusted. This adjustment is labeled by number 4 in the flowchart given by figure 4.1 and implemented using Python. Python was chosen for the implementation because it provides an easy way for reading CSV files and altering them, as well as providing the plotting library Matplotlib used for graphical evaluation of the data.

Figure 4.12 shows the adjustment of alternative feature groups necessary for SPL Conqueror for *fsTCAlgo*. The column for the parent feature, in this case *fsTCAlgo* itself is kept, but all values are changed to 1. In all models from simulation data used for this

Thesis any feature containing an alternative group is always mandatory. The adjustment can be expanded to also include optional features representing alternative groups. Each different option contained in the column of the original file is added as an additional column in the new file. As shown in figure 4.12, this results in additional columns for *MAXPOWER*, *DKTC*, and *EKTC* for the example provided. For each row the value of the newly created columns is 1, if the *fsTCAlgo* column from the original file contains its name in that row, otherwise its value is 0.

Row	fsTCAlgo	MAXPOWER	DKTC	EKTC	...
1	1	1	0	0	
2	1	1	0	0	
3	1	0	1	0	
4	1	0	1	0	
5	1	0	1	0	
6	1	0	1	0	
7	1	0	0	1	
...					

Figure 4.12: The table shows the result of an alternative group column being split into one column for each different entry.

For the models used to create the simulation data some of the topology control algorithms (TCAlgo) contain additional numeric features influencing their behavior. In the D4wsntraces simulation DKTC and EKTC require a k-value and L*KTC requires a k and an a-value, each represented by a numeric feature. SPL Conqueror requires one column for each of those numeric features. In the original data file (figure 4.11) the values for k (and a) are given in a single column, *fsTCParamK* (and *fsTCParamA*). If the row contains a TC algorithm which requires the numeric feature the value of it is given numerically, otherwise the value of the column for that row is given as *NaN*, meaning the parameter is not required (rows 1 and 2). For each parameter required by a TC algorithm an additional column is added, as shown in figure 4.13. The column is named with the parameter value and the name of the TC algorithm. For the k parameter of DKTC for example the column *kDKTC* is added. The value of that column is assigned 0 if the algorithm itself is not selected, otherwise the value given in the column *fsTCParamK* is used.

Row	...	DKTC	kDKTC	EKTC	kEKTC	...
1		0	0	0	0	
2		0	0	0	0	
3		1	1.0	0	0	
4		1	1.0	0	0	
5		1	2.0	0	0	
6		1	2.0	0	0	
7		0	0	1	1.0	
...						

Figure 4.13: The table shows the result of creating additional columns for TC algorithms requiring the additional parameter k.

SPL Conqueror does not support the definition of optional numeric features, as introduced in the previous section about the creation of the variability model. In figure 4.10 from the previous section an approach for creating an expanded equivalent of a numeric feature, by introduction of an additional optional binary feature, is shown. The CSV file containing the configuration information needs to be adjusted accordingly. For any optional numeric feature in the model used for the data set an additional column is introduced representing the optional selection of the feature. In figure 4.14 the column *WeightOptBinary* represents the added optional binary feature and *fsWeightOpt* in the right chart represents the actual numeric feature, which is now a mandatory child feature of *WeightOptBinary*. In rows 1 and 2 in the left chart the feature is not selected, therefore the optional binary feature column in the chart on the right contains a 0.

Row	...	fsWeightOpt	...
1		NaN	
2		NaN	
3		0.0	
4		20.0	
5		0.0	
6		20.0	
7		0.0	
...			

Row	...	WeightOptBinary	fsWeightOpt	...
1		0	0	
2		0	0	
3		1	0.0	
4		1	20.0	
5		1	0.0	
6		1	20.0	
7		1	0.0	
...				

Figure 4.14: Adjustment of an optional numeric feature for SPL Conqueror.

4.5 Generating an Influence Model using SPL Conqueror

SPL Conqueror supports two methods for generating the influence model: a GUI and a command line to run a-script files. The GUI requires each model and measurement file to be manually imported and allows the user to choose sampling strategies and other configurations for the learning process. The command line allows us to generate *a-scripts* automatically from the model and measurement files, define the learning process configurations in the scripts and execute them. For this thesis only the command line functionality is taken into consideration.

Listing 4.4 shows an example of an a-script file used for creating the influence model for the NFP energy consumption. The first line contains the log command which defines the output location. If the named file already exists it will be overwritten, otherwise a new log-file is created. The second line provides the command for loading the variability model from the given location. The third line provides the command for loading the measurements from the given location. SPL Conqueror supports two different data formats, an xml format and a CSV format. Since the simulation data created for development and evaluation is already in the CSV format, this thesis will only cover the usage of measurements in the CSV format. The fourth line is used for loading machine-learning settings (mlsettings). SPL Conqueror provides a wide range of settings for this option, including the maximum number of learning rounds (numberOfRounds) and the maximum number of features in an interaction (limitFeatureSize, featureSizeThreshhold). Line five specifies the NFP to be used for the learning algorithm, in this example the energy consumption. The name of the NFP given here must be contained as a column header in the loaded CSV file. Line six (learnwithallmeasurements) enables learning with all measurements provided in the measurements file. The command *analyze-learning* in line seven displays the learning results. If a log file location is specified as done in this example in line 1, the result will be printed to the specified file. The command *clean-sampling* in line eight is used to clear the sampling strategies. This can be used to specify multiple runs of learning for different sampling strategies using a single script.

Listing 4.4: Example of an a-script file used (e.g. EMeanScript.a) for the NFP energy consumption.

```
1 log C:\EMean.log
2 vm C:\vm.xml
3 all C:\measurements.csv
4 mlsettings numberOfRounds:15
5 nfp mEMean
6 learnwithallmeasurements
7 analyze-learning
8 clean-sampling
```

SPL Conqueror provides the command *script* for specifying subscripts. The *script* command can be used to run a batch of scripts. In this thesis we will be creating one script for each NFP measured in the measurement file and adding each of these scripts as subscripts to the final script which is executed in the command prompt. Listing

4.5 shows an example of a script containing subscripts. Each line contains the script command, followed by the location of the subscript. In the listing the subscripts define the learning settings for the NFPs energy consumption and different node lifetimes.

Listing 4.5: Example of an a-script containing subscripts.

```
1 script C:\EMeanScript.a
2 script C:\L25Script.a
3 script C:\L50Script.a
4 script C:\L75Script.a
```

Listing 4.6 shows an excerpt of the log file created when running the script introduced in listing 4.4. The file contains information about the commands executed, the influence model calculated and additional information, such as the learning error. Line two of the excerpt provides the header for the results of each round of the learning process, such as lines five and six. The first value gives the number of the round, line five shows the results for round one and line six shows the results for round two. The number of rounds calculated depends on the number specified in the script file or if no maximum of rounds is specified, a termination threshold for the learning error is used. For the implementation of the optimization model we are only interested in the influence model SPL Conqueror calculates, which is the second term of each round. For the first round the influence model is given by $99683708 * System + 404348 * aLSTARKTC$ (rounded to the nearest integer). The influence model given by SPL Conqueror functions like a formula containing all the relevant features and their respective influence on the value of the NFP. When entering the values of the features of a specific configuration into the influence model, the model will return the value of the NFP for that configuration, as it is approximated by the formula provided.

Listing 4.6: Excerpt of a log file created by SPL Conqueror for the NFP energy consumption.

```
1 command: analyze-learning
2 Round, Model, LearningError, LearningErrorRel, ValidationError,
      ValidationErrorRel, ElapsedSeconds, ModelComplexity, BestCandidate,
      BestCandidateSize, BestCandidateScore, TestError
3 Models:
4 Termination reason: numberOfRounds
5 1; 99683707.7502272 * System + 404348.466090221 * aLSTARKTC; 1.71880555813737;
      1.71880555813737; 1.71880555813737; 1.71880555813737; 0.1655016; 2;
      404348.466090221 * aLSTARKTC; 1; 1.79769313486232E+308; 1.71880555813737
6 2; 99551387.9683692 * System + 400510.674174517 * aLSTARKTC + 139561.162954614
      * MobSpeed; 1.71701748879584; 1.71701748879584; 1.71701748879584;
      1.71701748879584; 0.187; 3; 139561.162954614 * MobSpeed; 1;
      0.00178806934153086; 1.71701748879584
7 ...
8 15; ... + 39974.0810474179 * WeightOpt * TopologyDensity + ...
9 ...
10 Analyze finished
11 command: clean-sampling
```

The excerpt shown in listing 4.6 contains an influence model with feature interaction. Feature interactions are specified by a product of the value of influence for the given feature interaction and the features involved in the interaction. In line eight of the listing, round fifteen of the learning process contains a feature interaction, $39974 * WeightOpt * TopologyDensity$. The interaction between system feature *WeightOpt* and context feature *TopologyDensity* is assigned an influence value of 39974 (rounded to the nearest integer). For an interaction between two binary features, if both are selected the influence value is multiplied by one and added as part of the equation. If one of the features is not selected, than the value of that feature is set to 0 and the term containing that feature is also 0, since it is a product containing the feature value. For numeric features, as is the case with *TopologyDensity*, the value of the influence is actually dependant on the value of the numeric feature, since the value of the feature in the equation is not simply 0 or 1, but can take any value from within the range of the given feature.

4.6 Integrating the Influence Model into the Cardygan-ILP Model and solving for the optimal Configuration

The influence values assigned to features and interactions by the influence model from SPL Conqueror are used as weights in the objective weight function introduced in the previous chapter. The constraints confining the solution space, making up the first part of the optimization model, were previously translated from the FODA notation to a cardinality-based notation to the propositional logic. The missing part of the optimization model is now only the objective function. As previously introduced we want to use a weight function, containing decision variables and weights, for the objective function. The influence model contains both the influences of feature interactions and the influences some features have independently of other features.

In order to add the influence model and set the objective function, new decision variables for each term in the influence model are introduced. In figure 4.15 the translation from the influence model terms to the decision variables and weights is shown. The model used to develop this implementation and create synthetic interactions is the FM introduced in figure 3.3. The newly introduced decision variables are added to the previously defined propositional logic using CardyGAn-ILP as additional *BinaryVar*s. The influence model is read from the log file created by SPL Conqueror and split into a map containing the decision variables and the corresponding weights.

Influence Model Term	Decision Variable	Weight
110 * Maxpower	$Maxpower \rightarrow t_0$	$\omega_0 = 110$
40 * Pedestrian * Maxpower	$Pedestrian \wedge Maxpower \rightarrow t_1$	$\omega_1 = 40$
20 * Pedestrian * DKTC * TCFrequency	$Pedestrian \wedge DKTC \wedge TCFrequency \rightarrow t_2$	$\omega_2 = 20$

Figure 4.15: Assignment of decision variables and weights according to influence model.

The final step for enabling CardyGAn-ILP to solve for the optimal configuration is defining the objective function. In CardyGAn-ILP this is done by an *Objective* object which can be set to be either maximizing or minimizing. The object contains methods for defining the objective function as an arithmetic expression. In the case of the weight function utilized in this thesis the arithmetic expression for the objective function for the influence model given in figure 4.15 is given in equation 6. Since the feature *TCFrequency* is numeric the weight ω_2 is actually dependent on the value of the numeric feature.

$$ObjectiveFunction = \omega_0 * t_0 + \omega_1 * t_1 + \omega_2 * t_2 \tag{6}$$

Listing 4.7 shows the implementation of the objective function in CardyGAn-ILP. Line one sets up the *Objective* object *obj*, initialized with the parameter *false*, representing a minimization. Line two sets the arithmetic expression for the objective. The variables w_o, w_1, and w_2 are double values. The variables t_0, t_1, and t_2 are BinaryVars and *tcFrequency* is an IntVar. The classes used *sum*, *param*, and *mult* are provided by CardyGAn-ILP. Sum represents an addition of all paramters given. Param sets up the double value given as a parameter as a constant. Mult represents a multiplication of the parameters given. In the last multiplication the IntVar tcFrequency is added as a parameter since it represents a numeric feature and has an influence on the weight.

Listing 4.7: An example of the implementation of an objective function in CardyGAn-ILP.

```
1 Objective obj = model.newObjective(false);
2 obj.setExpr(sum(mult(param(w0), t0), mult(param(w1), t1), mult(param(w2), t2,
    tcFrequency)));
```

To evaluate if the proposed approach allows us to represent context dependent NFPs and derive optimal configurations in regard to these NFPs. For the evaluation and development of the proposed approach two simulation data sets were created. The first one is based on the FM introduced earlier in figure 3.1 and a second one shown in figure 5.1 below. A third set of data was created synthetically using the FM introduced in figure 3.3. The synthetic data was used to ensure the existence of feature interactions within the data by setting them manually. The synthetic data set is used to evaluate the result of the approach, since the expected outcome is known.

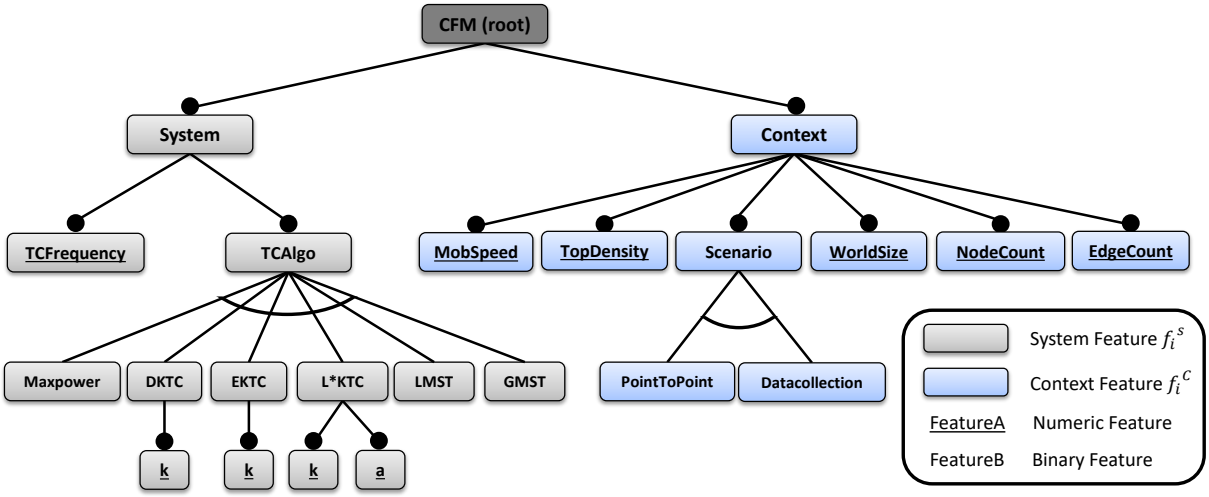

Figure 5.1: FM used for the second set of simulation data (D5wsntraces).

To evaluate the proposed approach the following set of research questions is examined:

- RQ1: Does our approach allow us to detect and represent feature interactions given an FM and measurements?
- RQ2: Is the approach still applicable if no context dependency between system features and context features exist?
- RQ3: Does the approach derive correct optimal configurations in regard of NFPs?
- RQ4: Is the approach practically applicable for realistic use cases with real world data?

5.1 RQ1: Detecting and representing feature interactions

Using SPL Conqueror for the creation of an influence model allows us to find and weight feature interactions using an FM and the associated measurements. By using the log file containing this influence model and the CardyGAn-ILP interface, we can represent the given feature interactions as propositional logic. Introducing a set of decision variables corresponding to the feature interactions allows the representation of these feature interactions as soft constraints. Any valid configuration can be selected using the hard constraints given through the FM definition and the soft constraints containing the weights can be used within this valid configuration to optimize the model. Using the synthetic data set mentioned before, in which feature interactions are set manually, we are able to confirm the correct representation of the feature interactions in the influence model and translated into decision variables.

5.2 RQ2: Applicability of the proposed approach in a model without context dependencies between system and context features

In order to examine the existence of feature interactions within the given simulation data sets a graphical analysis is performed first. Figure 5.2 shows the energy consumption measured as part of the first simulation (D4wsntraces) for different system configurations given on the x-axis. Each configuration selected in the simulation data is run over five different seeds, changing the initial composition of the topology. The standard deviation over the five seeds of each configuration is given as the error bars. In order to examine the effects of feature interactions on the NFP, the graphical analysis was done using fixed contexts. To create a set of fixed contexts, the numeric context features *MobSpeed* and *TopologyDensity* were split into two ranges, low and high. With the two ranges for each numeric feature and the three different *Scenario* options, the result is twelve different fixed contexts. A feature interaction represents an interaction between system and context features, a feature interaction present in the data would therefore result in a different measurement for the given NFP. In the figure shown however it can be seen, that independently of the system configurations each measurement is well within the error of all other measurements. Therefore we can assume that no feature interactions are present in the given data set.

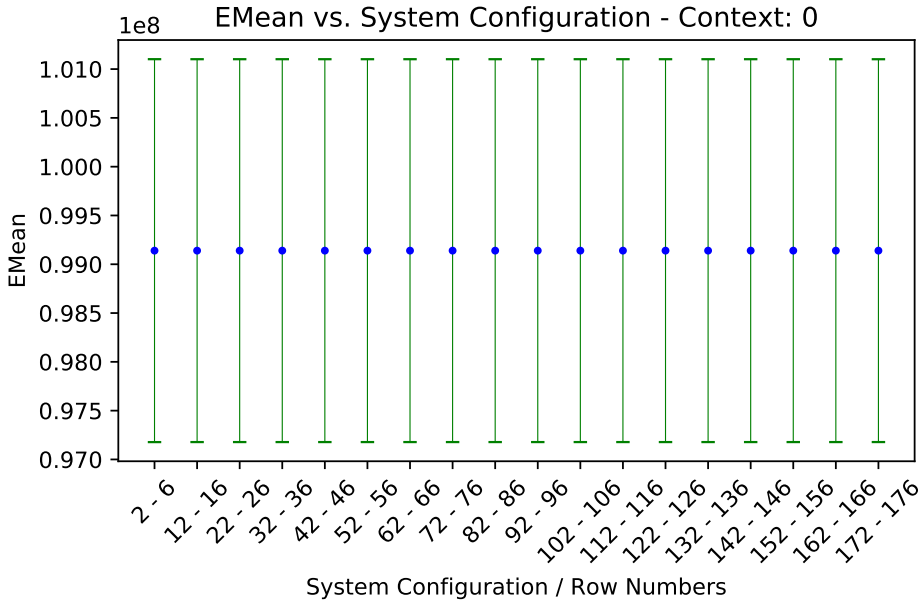

Figure 5.2: Energy consumption measurements for different system configurations and constant context.

Using SPL Conqueror we can create an influence model for the given simulation data. Listing 5.1 shows the influence model created for the NFP energy consumption. The last term of the listing represents a feature interaction between the system feature *WeightOpt* and the context features *TopologyDensity* and *MobSpeed*. However when compared to the base value of the NFP given by the mandatory selection of the *System* feature, the influence even for the maximum values of the numeric features (WeightOpt = 20, TopologyDensity = 20, MobSpeed = 1.5) is only 0.3% of the given base value. The proposed approach will still incorporate this minor feature interaction into the final optimization model.

Listing 5.1: Log file created for the NFP energy consumption.

```
97685634.4093499 * System +
102148.553907365 * aLSTARKTC +
-88839.5012020348 * MobSpeed +
724214.736813349 * MobSpeed * Datacollection +
51345.4703356032 * TopologyDensity * TopologyDensity +
-2706.78043369286 * TopologyDensity * TopologyDensity * TopologyDensity+
-503.958158623351 * WeightOpt * TopologyDensity * MobSpeed
```

Since the optimization of energy consumption is a minimization, the optimization model will select as few terms containing positive values and as many terms containing negative values. Since the influence model has assigned the a-paramter of the L*KTC algorithm a positive value independently of any context, the optimal configuration will never include the feature *aLSTARTKTC* and therefore, by definition of the FM, L*KTC is never part of an optimal configuration. The term containing the feature *WeightOpt* represents a feature

interaction and is assigned a negative value. Therefore any context configuration in which the MobSpeed is not equal to zero will contain the feature *WeightOpt* and the optimization model assigns it the maximum value from within the range, resulting in the largest negative value and thereby positive influence (due to minimization) on the NFP. Therefore we can confirm that even for no feature interactions, or insignificantly small feature interactions, the optimization model still creates conservative soft constraints confining the valid configuration space further and thus picking the most optimal configuration.

5.3 RQ3: Correctness of the optimal configuration derived from the proposed optimization model

To evaluate the correctness of the optimal configuration obtained from the optimization model we will be using the synthetic data set. The feature interactions were chosen for the NFP end-to-end droprate using weights representative of each interactions relative influences to each other but not of the value the NFP would actually take. The influence model for the synthetic data is given in listing 5.2.

Listing 5.2: Influence model of the synthetic data.

```
300 * System +
-20 * LMST +
-1 * WeightOpt +
-10 * Maxpower * Pedestrian * TopologyDensity +
-5 * DKTC * TCFrequency * Pedestrian
```

Using the influence model and the proposed approach for finding the optimal configuration we can produce a set of context configurations and find the optimal system configuration for that context. The optimal configuration for the end-to-end droprate has the minimal value, thus a minimization is required. Figure 5.3 shows a table containing the optimal configurations derived from the optimization model based on a give context. The first row contains the optimal system configuration for the context configuration pedestrian, *TopologyDensity* less than 12 and any Scenario. In figure 5.4 the influence model is shown graphically for the fixed context selection containing pedestrian and the varying of *TopologyDensity* from its minimum value to its maximum value. The graph confirms the result of the optimization model. For any value of *TopologyDensity* less than 12, the line marking the system selection *DKTC*, *TCFrequency* equal to twenty, and *WeightOpt* equal to twenty, represents the system configuration with the best influence on the NFP. For *TopologyDensity* values greater than 12 the system selection of *Maxpower* provides the optimal influence value. The optimization model correctly handles the constraints defined using the propositional logic definition of the FM by not choosing the *WeightOpt* feature, which is excluded by *Maxpower*, but would have an even greater influence on the NFP.

Context-Configuration	System-Configuration
Pedestrian, TopologyDensity < 12	DKTC, TCFrequency = 20, WeightOpt = 20
Pedestrian, TopologyDensity > 12	Maxpower
Static, (any TopologyDensity and Scenario selection)	LMST, WeightOpt = 20

Figure 5.3: Configuration results from the optimization model for different contexts.

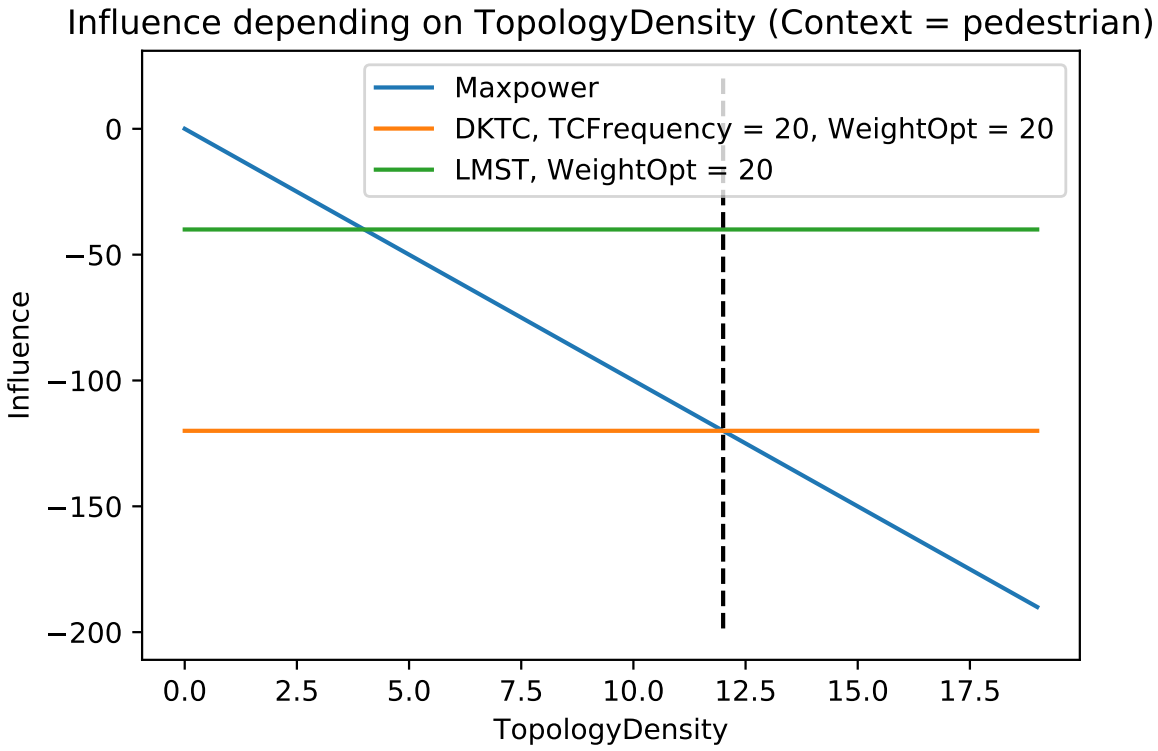

Figure 5.4: Graph showing the influence on the NFP end-to-end droprate depending on the TopologyDensity.

5.4 RQ4: Practical applicability of the proposed approach for real world data

We used two sets of simulation data, one containing 1180 and the other 7280 measurements, for developing, testing and evaluating the proposed approach. We were able to determine context dependencies in the form of feature interactions from the simulated data sets and enrich the hard constraints defining the FM with additional soft constraints describing the optimal configurations in regard to NFPs. The simulation data used did not contain any significant feature interactions, which resulted in a final system with very little necessary configuration adaptions and very insignificant influence of the selected system configuration on the NFP. The results of the simulation data analysis were verified with expert interviews and are deemed statistically plausible. As shown in the previous sections however we were still able to account for these minor feature interactions in

the most optimal way using the proposed approach. Therefore the approach allows for the aggregation of large amounts of data and deriving the model adaptions necessary to enable a system to configure optimally at runtime in regard to NFPs and context. In order to further confirm the results of the developed approach, additional development on the algorithms used for the simulation data or the usage of a different case study are necessary.

6 Conclusion

We proposed and developed an approach for optimizing an FM as part of a DSPL in regard to the influence of context-dependent features on NFPs in this thesis. The complete approach was implemented using the CardyGAn Tool Suite and SPL Conqueror, as well as Java, and Python. In the evaluation we were able to determine the correctness of the resulting model in regard to the general constraints provided by FMs. We were also able to show that the configuration derived is actually the optimal one in regard to a given context and NFP. Using simulation data of a communication network topology and different topology control algorithms we were showcased the applicability of the proposed approach on large amounts of real world data. The final workflow developed allows for the calculation of the optimal configuration adaptions of a DSPL given a single definition of the FM as a CardyGAn-Model model and the measurements of the NFP.

6.1 Future Work

In this thesis we only consider the optimal configuration in regards to the given context. The process developed can be expanded to include the reconfiguration cost. The reconfiguration cost is the potential resource cost to reconfigure a system. To expand the model by the reconfiguration cost, additional weights can be introduced, to model this cost. The value of the weights can be dependant on the previous configuration. This means that a certain interaction can exist between currently selected features and features contained in the optimal configuration during reconfiguration in regards to a certain resource (NFP). While some NFPs can be time dependant, meaning that their value accumulates over time, the reconfiguration is time independent. An example would be the energy consumption. A high energy consumption cost while reconfiguring could be compensated by a lower energy consumption for the new configuration over time. Thus an examination of the time independence of the reconfiguration must be considered when setting the weight.

References

[ALHM+11] Alférez, Mauricio ; Lopez-Herrejon, Roberto E. ; Moreira, Ana ; Amaral, Vasco ; Egyed, Alexander: *Supporting Consistency Checking between Features and Software Product Line Use Scenarios*. 2011

[CBT+14] Capilla, Rafael ; Bosch, Jan ; Trinidad, Pablo ; Ruiz-Cortés, Antonio ; Hinchey, Mike: An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. In: *Journal of Systems and Software* 91 (2014), S. 3–23

[KSV+17] Kluge, Roland ; Stein, Michael ; Varró, Gergely ; Schürr, Andy ; Hollick, Matthias ; Mühlhäuser, Max: A systematic approach to constructing families of incremental topology control algorithms using graph transformation. In: *Software & Systems Modeling (SoSyM)* (2017), März, 1–41. http://tubiblio.ulb.tu-darmstadt.de/85571/. – ISSN 1619–1374

[PBL05] Pohl, Klaus ; Böckle, Günter ; Linden, Frank van d.: *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005

[Pet90] Peterson, Kyo C. Kang; Sholomon C. Cohen; James A. Hess; William E. Novak; A. S.: *Feature-Oriented Domain Analysis (FODA) - Feasibility Study*. Carnegie-Mellon University Software Engineering Institute, 1990

[SGAK15] Siegmund, Norbert ; Grebhahn, Alexander ; Apel, Sven ; Kästner, Christian: *Performance-Influence Models for Highly Configurable Systems*. 2015

[Sie] Siegmund, Norbert: *SPL Conqueror: Measuring and Predicting Non-Functional Properties of Highly Configurable Software Systems*. http://www.infosun.fim.uni-passau.de/se/projects/splconqueror/, . – Accessed: 05.09.2017

[SKK+12] Siegmund, Norbert ; Kolesnikov, Sergiy S. ; Kästner, Christian ; Apel, Sven ; Batory, Don ; Rosenmüller, Marko ; Saake, Gunter: Predicting Performance via Automated Feature-Interaction Detection. (2012), S. 167–177

[SLR13] Saller, Karsten ; Lochau, Malte ; Reimund, Ingo: *Context-Aware DSPLs: Model-Based Runtime Adatpion for Resource-Constrained Systems*. 2013

[SRK+11] Siegmund, Norbert ; Rosenmüller, Marko ; Kuhlemann, Martin ; Kästner, Christian ; Apel, Sven ; Saake, Gunter: SPL Conqueror: Toward optimization of non-functional properties in software product lines. (2011), S. 487–517

[SWK+16] Schnabel, T. ; Weckesser, M. ; Kluge, R. ; Lochau, M. ; Schürr, A.: *CardyGAn: Tool Support for Cardinality-based Feature Models*. 2016. –

In Proceedings of the 10th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS), ACM

[WLS+16] Weckesser, Markus ; Lochau, Malte ; Schnabel, Thomas ; Richerzhagen, Björn ; Schürr, Andy: *Mind the Gap! Automated Anomaly Detection for Potentially Unbounded Cardinality-based Feature Models*. 2016