Feature Model Mining

by

Steven She

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2008

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Software systems have grown larger and more complex in recent years. Generative software development strives to automate software development from a systems family by generating implementations using domain-specific languages. In current practice, specifying domain-specific languages is a manual task requiring expert analysis of multiple information sources. Furthermore, the concepts and relations represented in a language are grown through its usage. Keeping the language consistent with its usage is a time-consuming process requiring manual comparison between the language instances and its language specification. Feature model mining addresses these issues by synthesizing a representative model bottom-up from a sample set of instances called configurations.

This thesis presents a mining algorithm that reverse-engineers a probabilistic feature model from a set of individual configurations. A configuration consists of a list of features that are defined as system properties that a stakeholder is interested in. Probabilistic expressions are retrieved from the sample configurations through the use of conjunctive and disjunctive association rule mining. These expressions are used to construct a probabilistic feature model.

The mined feature model consists of a hierarchy of features, a set of additional hard constraints and soft constraints. The hierarchy describes the dependencies and alternative relations exhibited among the features. The additional hard constraints are a set of propositional formulas which must be satisfied in a legal configuration. Soft constraints describe likely defaults or common patterns.

Systems families are often realized using object-oriented frameworks that provide reusable designs for constructing a family of applications. The mining algorithm is evaluated on a set of applications to retrieve a metamodel of the Java Applet framework. The feature model is then applied to the development of framework-specific modeling languages (FSMLs). FSMLs are domain-specific languages that model the framework-provided concepts and their rules for development.

The work presented in this thesis provides the foundation for further research in feature model mining. The strengths and weaknesses of the algorithm are analyzed and the thesis concludes with a discussion of possible extensions.

Acknowledgements

I would like to begin by thanking my supervisor, Professor Krzysztof Czarnecki, for providing the direction and guidance that has led to the completion of my Master's degree. His mentoring has taught me to enjoy the occasionally arduous, but ultimately rewarding process of research. Without his guidance, this thesis would not have been possible.

I would also like to thank Professor Joanne Atlee and Professor Michael Godfrey of the University of Waterloo, and Professor Andrzej Wąsowski of the IT University of Copenhagen for taking their time to read and suggest improvements to this thesis. I am grateful for their professional advice and helpful comments towards this work.

Furthermore, I would like to express thanks to my course instructors, Professor Ric Holt, Professor Steve MacDonald, and Professor Chrysanne Di Marco for introducing me to the world of research. Their knowledge and expertise have provided me with the academic background that has been invaluable for the completion of this thesis.

My colleagues in the Generative Software Development Lab have made it a great place to work. Our frequent discussions have helped me find my course of research and provided the much needed laughs during my studies.

I am grateful to my friends for being able to put up with my busy schedule and occasional absences to social gatherings. During my Master's, they have reminded me to take a break when it was the last thing on my mind.

I am indebted to my girlfriend, Anita, for having the patience to listen to my jargon filled presentations, erratic graduate student hours and for embracing my inner geekness. I am grateful that she has taken the time to complete the strenuous task of proof-reading and editing this thesis (which included this sentence). She has cared for me when I couldn't care for myself. Her love and support has provided me with inspiration and motivation.

Finally, I would like to express my sincerest appreciation to my family for their endless love, encouragement and support. They have taught me to keep an open-mind and to always strive to reach my full potential. With that, I was able to discover my strength in the field of Computer Science. Their unwavering faith in my abilities have been my motivation for seeking out this Master's degree and for excelling in my future studies.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Inevitably, software systems grow in both size and complexity. Object-oriented programming has greatly increased a developer's ability to model large and complex domains, however, object-orientation alone has been insufficient for achieving the scalability necessary for software systems today and in the future.

Generative software development adopts a systems engineering approach and automates the creation of software systems from a systems family by generating an implementation from a set of specifications written using domain-specific languages [Cza04]. In current practice, these languages are written in a top-down fashion by an expert through analysis of domain artifacts such as documentation and existing applications. In addition, languages are extended and grown over time by its usage. Incorporating new abstractions introduced by a language's usage is important to its evolution [Ste99]. This is particularly applicable in the area of generative software development where users construct domain-specific languages tailored to their specialized domain.

A systems family is a set of software systems that share a common infrastructure and product characteristics called *features*. Feature modeling is a key technique for identifying common and variable features in a systems family and formalizing such analysis in the form of a feature model. In the current state of the art, feature models are constructed using a top-down approach involving analysis of domain artifacts by an expert. Examples of feature models constructed using such an approach include template libraries [CE00], Telecom systems [LKL02], and e-commerce systems [Lau06]. *Feature model mining* introduces a novel bottom-up approach for retrieving a representative feature model from a sample set of system family members, called *configurations*.

Systems families are often realized using object-oriented frameworks. An object-oriented framework provides a reusable set of abstractions for building a family of framework applications. Frameworks have a set of rules for development, called its rules of engagement, which must be followed when building framework applications. Framework-specific modeling languages (FSMLs) are domain-specific languages for formalizing the framework-provided concepts in an object-oriented framework and its rules of engagement

[AC06]. FSMLs use feature models to describe a framework's rules and concepts. Thus, FSMLs offer the perfect target for applying feature model mining to a practical problem in software engineering.

Currently, building an FSML is a manual process that requires a framework expert to construct a feature model from several sources of information such as existing applications, developer documentation, tutorials and web articles. Feature model mining automates the construction by reverse-engineering a feature model from a sample set of applications. In this thesis, feature model mining is applied as a means of comparing framework applications with an FSML, discovering implementation patterns, and jump-starting FSML development.

The feature model mining algorithm is used to reverse-engineer a feature model for the Applet FSML [AC07]. Through the exploratory case study, several issues are identified and potential solutions are discussed. Consequently, the work presented in this thesis also suggests significant future work for improving and extending the mining algorithm.

1.1 Research Contributions

The following novel contributions are made in this thesis:

- The feature model mining algorithm which reverse-engineers a probabilistic feature model from a sample set of configurations is presented.
- A method for constructing disjunctive association rules using minimal OR-clauses is described.
- Applying feature model mining towards the development of an framework-specific modeling language (FSML) is described. The mined feature model is used to analyze framework usages and to refine an existing FSML.
- The work presented in this thesis provides the foundation for further research in the area of feature model mining. Short-term and long-term extensions and improvements to the mining algorithm are presented.

1.2 Notation

A negated feature i is denoted as $\bar{i}$ when describing propositional formulas. This thesis applies association rule mining in retrieving probabilistic propositional formulas. To differentiate association rules and Boolean implications, an association rule between two expression A and B is denoted by a double right arrow, $A \Rightarrow B$, while Boolean implications are denoted by a single right arrow, $A \rightarrow B$. The differences between association rules and implications are further discussed in Section 3.3.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 introduces fundamental concepts used in this thesis, such as feature models, data mining and framework-specific modeling languages (FSMLs). Chapter 3 presents the feature model mining algorithm and methodology in detail. Chapter 4 discusses several use cases of feature model mining for FSML development. Chapter 5 discusses extensions and improvements to the mining algorithm. Finally, Chapter 7 summarizes the work presented and concludes the thesis.

CHAPTER 2

Background

In this chapter, an overview of the areas that this thesis addresses is presented. In Section 2.1, basic feature models are introduced. Section 2.2 describes probabilistic feature models which are an extension of basic feature models with support for soft constraints. Section 2.3 briefly describes the necessary data mining concepts and terminology and finally, Section 2.4 provides a brief overview of framework-specific modeling languages (FSMLs).

2.1 Basic Feature Model

A *feature model* [KCH+90] consists of a feature hierarchy describing its variability and an optional set of constraints. A *feature* is a property that is relevant to some stakeholder [CE00]. For example, features in a car may be a gearbox, wheels, and an engine. A feature model can be instantiated in the form of a configuration. A *configuration* is a set of selected features according to the semantics of a feature model. Consequently, a feature model describes a set of *legal configurations* with respect to a set of features. A *basic feature model* consists of a set of hard constraints, separated into two components: (i) a feature hierarchy and (ii) a set of additional hard constraints. A *hard constraint* is a constraint in the feature model that must be satisfied in all legal configurations.

The feature hierarchy is a tree of features. A feature may have one or more child features called subfeatures. The top most feature in the feature hierarchy is called the root feature. In a basic feature model, all features other than the root feature must either be a *solitary* or a *grouped feature* [CE00]. Solitary features are either mandatory or optional. Mandatory features should be selected in all legal configurations, and optional features may or may not be selected provided that its parent is selected. Grouped features are subfeatures of either an OR-group (inclusive-or) or a XOR-group (exclusive-or). An OR-group requires that at least one subfeature be selected in a legal configuration. A XOR-group requires that exactly one subfeature must be selected in a legal configuration.

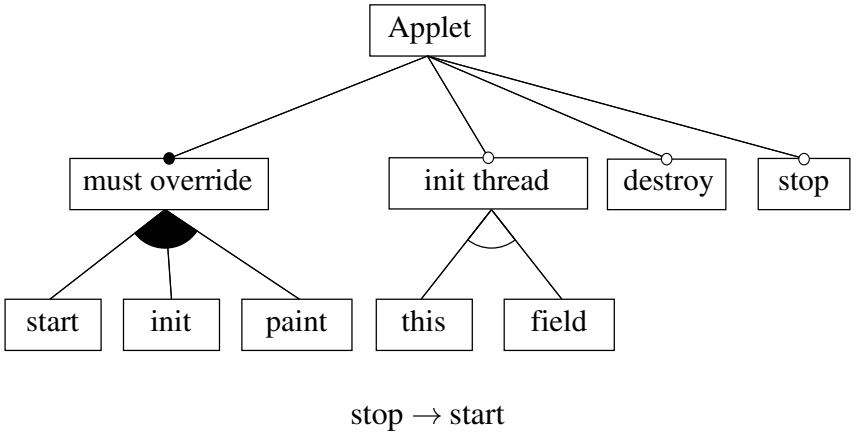


Figure 2.1: Basic feature model of an applet

In additional to the feature hierarchy, a basic feature model may have an additional set of hard constraints. Additional hard constraints are specified as a list of propositional formulas. These additional hard constraints further limit the set of legal configuration as described by the feature hierarchy.

Feature modeling has also been extended with support for other concepts, such as feature and group cardinalities, attributes and references [CHE05]. These extensions are not covered by the mining approach presented in this thesis, but can be solved as part of future work.

In this thesis, two graphical renderings of feature models are used: (i) a feature diagram, and (ii) a textual representation.

The feature diagram notation was first introduced by Kang et al. in their Feature Oriented Domain Analysis (FODA) [KCH+90] methodology. It was later expanded upon by Czarnecki and Eisenecker with the addition of OR-groups [CE00]. In this notation, features are represented by labeled rectangular boxes. The root feature is represented as the top most, superior feature with no ancestors. Subfeatures are indicated by a line connecting parent to child. Solitary features are denoted by a small circle where a filled circle indicates a mandatory feature and an empty circle denotes an optional feature. Grouped features are represented by an arc between its subfeatures. A filled arc represented a OR-group, while an empty arc represents an XOR-group. The list of additional hard constraints is shown below the feature hierarchy.

A second, textual rendering of a feature model is also used in this thesis. The notation is based on the tree notation used in the Eclipse Feature Modeling Plug-in (fmp) [CAK+05]. In a textual representation, descendants of a feature are indicated by its indentation. The topmost feature in the tree is the root feature and is shown in **bold face**. Indentation is used to indicate subfeatures. A solitary feature is prefixed with its cardinality, where $[0..1]$ represents an optional feature and $[1..1]$ represents a mandatory feature. An OR-group with k features is shown as $\langle \mathbf{1}\text{–}\mathbf{k} \rangle$, and an XOR-group is shown as $\langle \mathbf{1}\text{–}\mathbf{1} \rangle$. Additional constraints are listed below the feature hierarchy.

In Figure 2.1, a basic feature model of an Applet is shown as a feature diagram. In this example, Applet is the root feature. The features must override, init thread, destroy, and stop are solitary features and subfeatures of Applet. The must override node is an OR-group, with paint, start, and init as its subfeatures. init thread is an XOR-group. The equivalent textual representation of this feature model is shown in Figure 2.2.

2.1.1 Connection with Propositional Logic

The constraints imposed by a basic feature model can be represented as a conjunction of propositional formulas [Bat05, BBC06]. The features in a feature model are translated to a set of Boolean variables. Each feature and feature group in the feature hierarchy of a basic feature model can be systematically broken down into the following hard constraints:

Child-parent implications. The nesting relations in the feature hierarchy are represented as implications from a child feature to its parent feature. The following form of constraint models the nesting for a parent feature f_p with a subfeature f_c:

$$f_c \rightarrow f_p \tag{2.1}$$

Mandatory features. A mandatory feature is a feature that must be present in all legal configurations given the presence of its parent feature. Translating this constraint into a propositional formula, mandatory features are bi-implications between parent and child. The bi-implication provides the constraint that the presence of the parent implies the presence of the child, and vice versa. In addition to the child-parent implication shown in Equation 2.1, A mandatory subfeature f_c with parent feature f_p has the following constraint:

$$f_p \rightarrow f_c \tag{2.2}$$

AND-groups. An AND-group is a set of features that are always present given the presence of its parent feature. An AND-group is logically equivalent to a set of mandatory features with the same set of ancestors. An AND-group with features $f_1, \ldots, f_k$ has the following constraint:

$$\bigwedge_{i=1}^{k} \bigwedge_{\substack{j=1 \\ i \neq j}}^{k} f_i \rightarrow f_j \tag{2.3}$$

OR-groups. The OR-group requires one or more subfeatures to be selected in a legal configuration. Thus, a constraint exists from the parent to the subfeatures in the form of an OR-clause. Expressing an OR-group f with subfeatures $f_1, \ldots, f_k$ in propositional logic requires adding the following constraint in addition to the child-parent implications:

$$f \rightarrow \bigvee_{i=1}^{k} f_i \tag{2.4}$$

Applet
- $[1..1]$ must override
  - $\langle$**1–3**$\rangle$
    - $[0..1]$ paint
    - $[0..1]$ start
    - $[0..1]$ init
- $[0..1]$ init thread
  - $\langle$**1–1**$\rangle$
    - $[0..1]$ this
    - $[0..1]$ field
- $[0..1]$ destroy
- $[0..1]$ stop

Additional Constraints:

stop $\rightarrow$ start

Figure 2.2: Textual rendering of the applet feature model

(a) child-parent implications

$$\begin{aligned}
\text{stop} &\rightarrow \text{Applet} \\
\text{destroy} &\rightarrow \text{Applet} \\
\text{init thread} &\rightarrow \text{Applet} \\
\text{must override} &\rightarrow \text{Applet} \\
\text{paint} &\rightarrow \text{must override} \\
\text{start} &\rightarrow \text{must override} \\
\text{init} &\rightarrow \text{must override} \\
\text{this} &\rightarrow \text{init thread} \\
\text{field} &\rightarrow \text{init thread}
\end{aligned}$$

(b) parent-child and group implications

$$\begin{aligned}
\text{Applet} &\rightarrow \text{must override} \\
\text{must override} &\rightarrow \text{paint} \vee \text{start} \vee \text{init} \\
\text{init thread} &\rightarrow \text{this} \vee \text{field}
\end{aligned}$$

(c) mutual exclusion conditions

$$\begin{aligned}
\text{this} &\rightarrow \overline{\text{field}} \\
\text{field} &\rightarrow \overline{\text{this}}
\end{aligned}$$

Figure 2.3: Propositional formulas representing the applet feature model

XOR-groups. The XOR-group is a specialization of the OR-group with the additional constraint of mutual exclusion amongst it's subfeatures (ie. one and only one feature may be selected in the XOR-group). Consequently, an XOR-group f with subfeatures $f_1, \ldots, f_k$ require the constraints of an OR-group and the following set of mutual exclusion clauses:

$$\bigwedge_{i=1}^{k} \bigwedge_{\substack{j=1 \\ i \neq j}}^{k} f_i \rightarrow \overline{f_j} \tag{2.5}$$

A feature model is logically represented by its *feature model formula* which is the conjunction of the propositional formulas for features, feature groups and additional constraints. Thus, a legal configuration corresponds to a valuation of the feature model formula that evaluates to *true*.

2.2 Probabilistic Feature Model

Probabilistic feature models (PFMs) introduces a new constraint called the soft constraint to basic feature models by using probabilistic logic constraints [CSW08]. A *soft constraint* is a constraint that should be satisfied by *most*, but not necessarily all legal configurations. As a result, a legal configuration may violate a soft constraint and still remain legal. Soft constraints can be used to specify defaults or preferences in a PFM. This is in contrast to a hard constraint, which is a constraint that must be satisfied in all legal configurations. Soft constraints have no impact on individual configurations, however, the set of soft constraints models a distribution of configurations for a PFM.

There is no single formalism for specifying soft constraints in a PFM. For example, we could choose a loose description of preference, such as using the terms "encourages" and "discourages". In the case of the feature model mining algorithm, the mined PFM uses conditional probabilities in the form of *confidence*.

In Figure 2.4, a probabilistic feature model of an applet is shown. In the probabilistic feature model, the constraint $stop \rightarrow start$ is modeled as a soft constraint instead of a hard constraint as in the basic feature model in Figure 2.1. The soft constraint provides the user with the information that if *stop* was selected in a configuration, then *start* should probably be selected as well. However, modeling the relationship as a soft constraint allows an individual configuration to violate the particular constraint and still be a legal configuration.

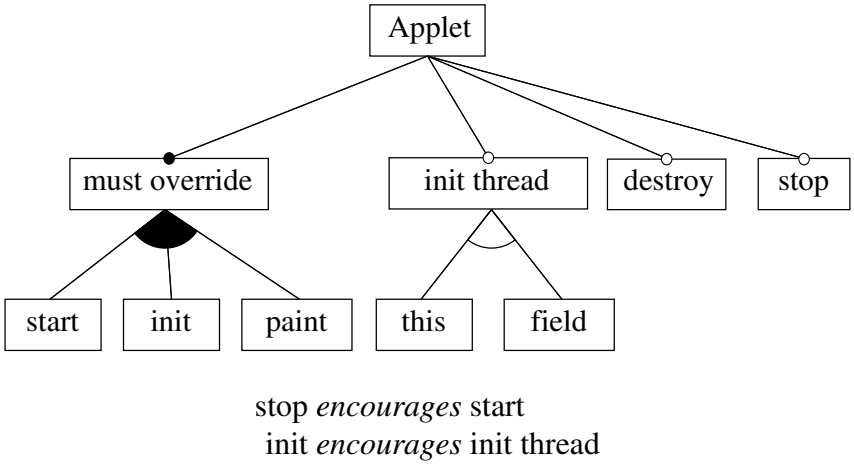


Figure 2.4: Probabilistic feature model of an applet

2.3 Data Mining

Data mining is the process of extracting knowledge from large amounts of data [HK00]. Feature model mining uses *association rule mining* to retrieve the necessary propositional formulas to construct a probabilistic feature model. In this section, data mining concepts will be related to feature model terminology and an overview of association rule mining follows.

A sample set of configurations, $\mathcal{S}$, and a set of features, $\mathcal{F}$, is provided as input into the feature model mining algorithm. A sample set contains a multiset of sample configurations. A sample configuration, $c \in \mathcal{S}$, is a set of selected features from $\mathcal{F}$. Each sample configuration has an associated id, called a *tid* in order differentiate configurations in the multiset. The absent features in a configuration, c, is represented as $\overline{c}$.

In data mining terminology, a sample set is equivalent to a dataset. A sample configuration is the same as a transactions and a feature is equivalent to an item or attribute in data mining literature. Since this thesis is mainly concerned with feature models, the feature model terminology will be primarily used.

The two notations for representing a sample set used in this thesis are shown in Table 2.1 on the following page. At the top of the table, the set of features, $\mathcal{F}$, is shown. In Table 2.1a, the sample set is represented as a *list* of configurations. In the left column, the tid is shown, and in the right, the set of selected features for the configuration is shown. In Table 2.1b, the *tabular form* for a sample set is presented. Here, the columns consist of the set of features, $\mathcal{F}$. A check mark (✓) in a column represents that the feature is selected in a configuration.

Association rule mining is used by the mining algorithm to retrieve propositional formulas from the sample set. An *association rule*, written $A \Rightarrow B$, is an expression between two Boolean formulas A and B with an associated *interestingness*. The interestingness measures the quality and strength of the association rule. The measures of interesting-

$$\mathcal{F} = \{\text{Override}, \text{Start}, \text{Init}, \text{Paint}\}$$

id	Configuration
1	Override Init Paint
2	Override Start Init
3	Override Start

(a) List of configurations

Tid	Override	Start	Init	Paint
1	✓		✓	✓
2	✓	✓	✓	
3	✓	✓		

(b) Tabular form

Table 2.1: Sample set notations

ness used in this thesis are *support* and *confidence*. Support is a measure of statistical significance and confidence is a measure of rule strength. An association rule with 100% confidence is equivalent to a boolean implication. A detailed definition of support and confidence is provided in Section 3.3.

A significant but subtle difference between an association rule and a Boolean implication exists. An association rule was defined by Agrawal et al. as an implication between two expression [AIS93]. During the course of research, it was found that a Boolean implication and an association rule have a different interpretation when the confidence of an association rule is less than 100%. An association rule can model uncertainty when its confidence is less than 100%. A Boolean implication cannot model such a concept. An association rule is equivalent to a Boolean implication only in the case where the rule has a confidence of 100%. The differences are further explained in Section 3.3.

2.4 Framework-Specific Modeling Languages

We apply feature model mining to the development of framework-specific modeling languages (FSMLs). An FSML is a domain-specific language that describes concepts in an object-oriented framework [AC06]. Framework-provided concepts and relations are represented in a metamodel specified using a feature model. A given FSML configuration is called a *framework-specific model* and it describes the concepts as implemented in the corresponding sample application [ABC07].

The metamodel of an FSML is an extended feature model enriched with support for attributes and references [CHE05]. The mining algorithm presented in this thesis is incapable of mining for these extension. This thesis is focused on mining FSMLs that use basic feature models. Implementing the mentioned extensions is left as future work.

We have identified the following use cases for feature model mining in the context of FSMLs: (i) as a basis for jump-starting development of an FSML, (ii) as an means of comparing an expert-specific FSML and sample applications, and (iii) as a method of refining an existing FSML by discovering new patterns in sample code. We discuss FSMLs and the application of feature model mining to these use cases in Chapter 4.

CHAPTER 3

Mining for Feature Models

The core of the feature model mining algorithm lies in the retrieval of propositional formulas for each type of feature in a basic feature model. The mined propositional formula with 100% confidence are conjoined to form the feature hierarchy formula. Next, a feature model synthesis algorithm introduced by Czarnecki and Wąsowski [CW07] is used to construct the mined feature model.

The chapter begins with an overview of the feature model mining algorithm in Section 3.1. The overview will provide a better understanding of the mining process using an example to demonstrate the steps of the algorithm from start to finish.

The connection between a sample set and propositional logic is described in Section 3.2. Two important operators are established, the *configuration expression* and the *supporting configurations*. The configuration expression defines a mapping from a configuration to a propositional formula, while the supporting configuration defines an inverse mapping.

Section 3.3 describes the use of association rules for feature model mining. Association rule mining is used to retrieve the various forms of propositional formulas from a sample set. Association rules describe a relationship between two expression and have a measure of quality. Support and confidence are the two measures used in this thesis to quantify statistical significance and the quality of the rules respectively. These measures, called *rule interestingness measures*, are further described in this section.

Section 3.4 presents an in-depth description of the feature model mining algorithm. Following the semantic constructs of feature models described in Chapter 2, three forms of implications are mined to construct a probabilistic feature model: (i) binary implications, (ii) group implications, and (iii) mutual exclusion clauses. The process and techniques used for retrieving the various forms of formulas for each feature type are described.

The mining algorithm uses the feature model synthesis algorithm by Czarnecki and Wąsowski to construct a feature model after having mined for the formulas in the sample set. The synthesis algorithm constructs a feature model using a propositional formula as input. The algorithm is described in further detail in Section 3.5.

The implemented mining algorithm is presented in Section 3.6. Finally, the chapter ends with a description of the current prototype in Section 3.7.

3.1 Overview of the Algorithm

Feature model mining retrieves a probabilistic feature model representative of a sample set of configurations. The mining algorithm relies on the property that feature models can be synthesized from propositional logic [CW07]. Thus, the goal of the feature model mining algorithm is to retrieve the propositional formula for the semantic constructs of a basic feature model as described in Section 2.1.1. These formulas are mined from the sample set using association rule mining. Association rules mined with 100% confidence are equivalent to implications and thus, are used to construct the feature hierarchy. This equivalence is further explained in Section 3.3. Rules with less than 100% confidence are presented to the user as a list of soft constraints.

Conceptually, the mining algorithm can be broken down into the following steps:

1. Mine for the following association rules:
   i. in the form of binary implications,
   ii. group implications,
   iii. and mutual exclusion clauses.
2. Create the feature hierarchy formula.
3. Construct feature model using the feature model synthesis algorithm.
4. Construct set of soft constraints.

The algorithm begins by mining for binary implications, which are used to construct the feature hierarchy and AND-groups. This step involves finding implications of the form $f_i \rightarrow f_j$ in the sample set. The implications are found using conjunctive association rule mining. Rules with 100% confidence are conjoined to form the feature hierarchy formula.

Next, group implications are mined in order to construct OR-groups. Group implications have the form $f \rightarrow f_1 \vee \cdots \vee f_k$. These constraints are found using disjunctive association rule mining. 100% confidence rules are selected and conjoined with the feature hierarchy formula.

The last form of expression is the mutual exclusion clause. These clauses are used to strengthen existing OR-groups into XOR-groups. Mutual exclusion clauses are implications of the form $f_i \rightarrow \overline{f_j}$, where $f_i, \ldots, f_j$ are subfeatures of an OR-group. If mutual exclusion clauses for all subfeatures in an OR-group are found in the feature hierarchy formula with 100% confidence, then the feature group becomes an XOR-group.

All mined association rules with less than 100% confidence are gathered to form the set of soft constraints.

The feature diagram is constructed using the feature model synthesis algorithm by Czarnecki and Wąsowski [CW07]. All mined rules with 100% confidence are conjoined together to form the feature hierarchy formula. The formula is fed into the synthesis

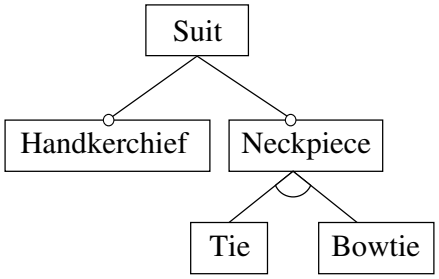
$$\mathcal{F} = \{\text{Suit}, \text{Handkerchief}, \text{Neckpiece}, \text{Tie}, \text{Bowtie}\}$$

id	Configuration
1	Suit
2	Suit, Handkerchief
3	Suit, Neckpiece, Tie
4	Suit, Neckpiece, Bowtie

(a) Sample set

$$\begin{aligned}
\text{Handkerchief} &\rightarrow \text{Suit} \\
\text{Neckpiece} &\rightarrow \text{Suit} \\
\text{Tie} &\rightarrow \text{Neckpiece} \\
\text{Bowtie} &\rightarrow \text{Neckpiece} \\
\text{Neckpiece} &\rightarrow \text{Tie} \vee \text{Bowtie} \\
\text{Neckpiece} &\rightarrow \overline{\text{Handkerchief}} \\
\text{Neckpiece} &\rightarrow \overline{\text{Tie}} \\
\text{Bowtie} &\rightarrow \overline{\text{Handkerchief}} \\
\text{Tie} &\rightarrow \overline{\text{Bowtie}} \\
\text{Tie} &\rightarrow \overline{\text{Handkerchief}}
\end{aligned}$$

(b) Mined association rules with 100% confidence



(c) Mined feature model

Figure 3.1: From a sample set of configurations to a mined model

algorithm which returns a feature diagram representing the inputted formula. The set of soft constraints consists of all remaining rules with less than 100% confidence.

As an example of the feature mining process, let us assume that a sample set is constructed from the sales data of four people at a suit store. The first person purchased just a suit with no additional items. The second person purchased two items: a suit and a handkerchief. The third purchased a suit and a tie which is categorized as a neckpiece in the system. The fourth purchased a suit and a bowtie (also considered a neckpiece). From the sales data, the set of features, $\mathcal{F}$, is: Suit, Handkerchief, Neckpiece, Tie and Bowtie. The resulting sample set is shown in Figure 3.1a.

Now, the feature mining algorithm is executed on the sample set. The mined formulas with 100% confidence are shown in Figure 3.1b. The first four implications shown are examples of binary implications. The fifth rule, $Neckpiece \rightarrow Tie \vee Bowtie$, is a group implication and is used to synthesize an OR-group between Tie and $Bowtie$. Finally, the last

five implications are the mutual exclusion clauses. Note that the mutual exclusion clauses are bi-directional, in that a mutual exclusion clause $x \rightarrow \overline{y}$ is equivalent to $y \rightarrow \overline{x}$. For example in the mined rules, $Tie \rightarrow \overline{Bowtie} \equiv Bowtie \rightarrow \overline{Tie}$. Only one mutual exclusion clause is useful to us, namely, $Tie \rightarrow \overline{Bowtie}$. This clause is used to strengthen the OR-group between *Tie* and *Bowtie* into an XOR-group.

The mined implications are conjoined together to form a *feature hierarchy formula*, which is inputted into the feature model synthesis algorithm. The resulting mined feature model is shown in Figure 3.1c. Suit is the root feature of the mined feature model. Handkerchief and Neckpiece are optional features of Suit. Neckpiece has an exclusive-or relationship between Tie and Bowtie.

3.2 Sample Sets and Propositional Logic

The sample set of configurations, $\mathcal{S}$, simply called the sample set for short is the set of configurations used for data mining. A sample configuration c is a set of selected features. The *negated configuration*, $\overline{c}$, consists of the features that were unselected, or absent from a configuration. The set of all features, $\mathcal{F}$, is the union of all features in the sample set. The definition of a configuration and its negated configuration is shown in Equation 3.2.

$$c \subseteq \mathcal{F} \tag{3.1}$$

$$\overline{c} = \mathcal{F} \setminus c \tag{3.2}$$

A mapping from configurations to propositional logic is first defined. The *configuration expression*, $\mathbf{r}(c)$, is a clause constructed from a conjunction of all selected features and a negation of all unselected features in a configuration:

$$\mathbf{r}(c) = \bigwedge_{v \in c} v \quad \wedge \quad \bigwedge_{w \in \overline{c}} \overline{w} \tag{3.3}$$

In the inverse direction, the *supporting configurations* of a Boolean expression, $\mathbf{c}(\alpha)$, is the set of configurations with configuration expressions that satisfy the boolean expression α:

$$\mathbf{c}(\alpha) = \{c \in \mathcal{S} \mid \mathbf{r}(c) \text{ satisfies } \alpha\} \tag{3.4}$$

The *sample set expression*, Φ, describes the set of all configurations in the sample set. The sample set expression, Φ, is an expression in disjunctive-normal form (DNF) constructed as the disjunction of all the configuration expressions in the sample set:

$$\Phi = \bigvee_{c \in \mathcal{S}} \mathbf{r}(c) \tag{3.5}$$

Tid	Itemset
c_1	A B C
c_2	B C

$\mathcal{F} = \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}\}$

(a) Sample set

$$\mathbf{r}(c_1) = A \wedge B \wedge C \wedge \overline{D} \wedge \overline{E}$$
$$\mathbf{r}(c_2) = \overline{A} \wedge B \wedge C \wedge \overline{D} \wedge \overline{E}$$

(b) Configuration expressions

$$\mathbf{c}(A \wedge B) = \{c_1\}$$
$$\mathbf{c}(B \wedge C) = \{c_1, c_2\}$$
$$\mathbf{c}(A \vee D) = \{c_1\}$$

(c) Supporting configurations

Figure 3.2: Configuration expressions and supporting configurations

Figure 3.2 is an example illustrating the defined operators in a sample set. The sample set, shown in Figure 3.2a, consists of two configurations, c_1 and c_2. The set of features, $\mathcal{F}$, consists of five features, A through E. Figure 3.2b is the set of configuration expressions for the sample set. Figure 3.2c shows several examples of supporting configurations. The sample set expression is the disjunction of all configuration expressions in the sample set, which is $\mathbf{r}(t_1) \vee \mathbf{r}(t_2)$.

3.3 Association Rules

The mining algorithm uses several forms of association rule mining to retrieve the necessary formulas. An association rule, $A \Rightarrow B$, is an expression between two propositional formulas, A and B with a degree of interestingness. Similar to Boolean implications, the expression A in the association rule is called the antecedent, and the expression B is known as the consequent. Association rules are directional, such that $A \Rightarrow B$ and $B \Rightarrow A$ are two distinct association rules.

The quality of an association rule is measured by its *interestingness*. An interestingness measure assigns a numeric value quantifying an association rule and allowing it to be interpreted and compared. The implemented algorithm uses two measures to judge the quality of a mined association rule, support and confidence [AIS93].

The *support* of an association rule is a measure of its statistical significance. Support is defined as the relative size of the set of supporting configurations for A and B with respect to the size of the sample set:

$$supp(A \Rightarrow B) = \frac{|\mathbf{c}(A \wedge B)|}{|\mathcal{S}|} \tag{3.6}$$

A configuration *supports* an association rule $A \Rightarrow B$ if and only if both A and B are satisfied in the configuration. The support of an association rule can also be interpreted

in probability theory. The support of association rule $A \Rightarrow B$ can be thought of as the probability of events A and B occurring together, which is written as $P(A \cap B)$.

Confidence measures the accuracy or strength of an association rule. In terms of supporting configurations, the confidence of an association rule is the size of the supporting configurations of $A \wedge B$ relative to the size of the satisfying configurations of A:

$$conf(A \Rightarrow B) = \frac{|\mathbf{c}(A \wedge B)|}{|\mathbf{c}(A)|} \tag{3.7}$$

Intuitively, an $x\%$ confidence for an association rule states that given the antecedent evaluates to $true$ for a configuration c, there is a $x\%$ chance that the configuration c also satisfies the consequent. Confidence can also be written in terms of support as shown in Equation 3.8.

$$conf(A \Rightarrow B) = \frac{supp(A \Rightarrow B)}{supp(A)} \tag{3.8}$$

In terms of probability theory, the confidence of an association rule $A \Rightarrow B$ is the same as the conditional probability of B given A, written $P(B|A)$ [HK00]. The definition of confidence in Equation 3.8 is analogous to the definition of $P(B|A)$, which is $P(A \cap B)/P(A)$.

In association rule literature, an association rule is written as $A \Rightarrow B$, and is often referred to as an implication [AIS93, AS94]. In our work, we treat an association rule, $A \Rightarrow B$ with 100% confidence as being equivalent to the implication $A \rightarrow B$ in the sense that the implication $A \rightarrow B$ holds across all configurations in the sample set.

An association rule, $A \Rightarrow B$ with 100% confidence is a statement that the presence of A entails the presence of B in the sample set, in all configurations that A is present in. As a result, it follows from the definition of confidence in Equation 3.7 that given an association rule with 100% confidence, the supporting configurations of $A \wedge B$ is the same as the supporting configurations of A:

$$\mathbf{c}(A \wedge B) = \mathbf{c}(A) \tag{3.9}$$

An implication $A \rightarrow B$ is equivalent to the expression $\overline{A} \vee (A \wedge B)$. Thus, the supporting configurations of an implication $A \rightarrow B$ with a corresponding 100% confidence association rule $A \Rightarrow B$ is:

$$\begin{aligned} \mathbf{c}(A \rightarrow B) &= \mathbf{c}(\overline{A}) + \mathbf{c}(A \wedge B) && (3.10)\\ &= \mathbf{c}(\overline{A}) + \mathbf{c}(A) && (3.11)\\ &= \mathcal{S} && (3.12) \end{aligned}$$

Thus, the implication constructed from an association rule with 100% confidence holds for all configurations in the sample set and will evaluate to $true$ for the sample set

Tid	Clock	Alarm	Digital	24hr	Analog	Hands
1	✓		✓	✓		
2	✓	✓	✓			
3	✓				✓	✓
4	✓	✓			✓	✓

Table 3.1: Sample set of clock configurations

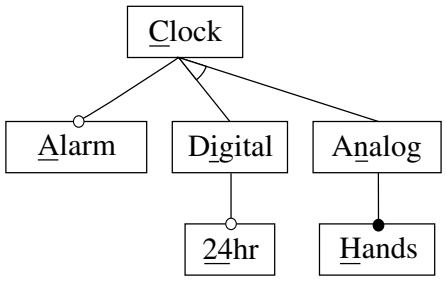


Figure 3.3: Expert-specified clock feature model

expression, Φ. As a result, an association rule $A \Rightarrow B$ with 100% confidence is equivalent to an implication $A \rightarrow B$.

However, in the case where the association rule has less than 100% confidence, there is no simple relation between the support of an association rule and that of an implication. Association rules have a range of uncertainty for which its interestingness quantifies, while Boolean implications do not, and must evaluate to $true$ or $false$ for each configuration. As a result, an association rule cannot be interpreted in terms of propositional logic.

3.4 Retrieving Feature Model Formulas

The core of the feature model mining algorithm lies in mining the propositional formulas for the feature types from a sample set. The various feature types in a basic feature model and their translation to propositional logic were introduced in Section 2.1. These feature types can be generalized into three different forms of implications: (i) binary implications, (ii) group implications, and (iii) mutual exclusion clauses.

Four clock configurations, shown in Figure 3.1, will be used as the example input sample set for the algorithm in this section. As a reference feature model, the clock configurations conform to the expert-specified feature model shown in Figure 3.3. Throughout the remainder of this section, each step the mining process will be demonstrated on this sample set.

In Section 3.4.1, techniques for retrieving binary implications are described. Section 3.4.2 presents disjunctive association rules and mining for group implications. Section 3.4.3 describes the process of mining for mutual exclusion clauses. The mining algorithm is also capable of mining for multiple feature models, which is described in Section 3.4.4. Techniques for selecting additional hard constraints are presented in Section 3.4.5. Finally, the mining process for the set of soft constraints is described in Section 3.4.6.

3.4.1 Binary Implications

Binary implications have the form $f_i \rightarrow f_j$ and are used to construct two types of features: (i) the feature hierarchy in the form of child-parent implications, and (ii) AND-groups as parent-child implications. AND-groups are logically equivalent to mandatory features. In order to separate an AND-group into a hierarchy of mandatory features, additional knowledge is required. This extension to the mining algorithm is discussed further in Section 5.4.2.

A binary implication $f_i \rightarrow f_j$ is equivalent to the following association rules with 100% confidence called *binary association rules*:

$$f_i \Rightarrow f_j \tag{3.13}$$

Thus, we mine for binary association rules using two alternative approaches:

1. *Feature combinations*. The first approach for retrieving binary association rules is by iterating through all feature tuples $(f_i, f_j) \in \mathcal{F}^2$ and creating an association rule $f_i \Rightarrow f_j$ for each tuple. This approach has the advantage of being fast and simple. While this method finds the necessary rules for constructing the feature hierarchy, soft constraints involving AND-groups are scattered over multiple binary binary association rules.

2. *Conjunctive association rules*. This approach mines for *conjunctive association rules* which have the form shown in Equation 3.14. Binary association rules are a subset of conjunctive association rules, however, conjunctive association rules have an added advantage that soft constraints involving AND-groups are codified as a single association rule.

$$\bigwedge_{x \in X} x \Longrightarrow \bigwedge_{y \in Y} y \tag{3.14}$$

Mining for conjunctive association rules can be separated into two steps [HK00]. First, frequent itemsets are mined from the sample set. Afterwards, the association rules are constructed using the mined frequent itemsets.

A *frequent* itemset is a set of features that satisfy a minimum user-specified support. As a sample set grows in size, the number of frequent itemsets can grow to be very large [UAUA04]. As a result, the mining procedure mines for either maximal or closed itemsets.

Two common forms of frequent itemsets are used in practice, maximal and closed frequent itemsets. A *maximal* itemset is a set of features such that no proper superset of features is also frequent. A *closed* itemset is a set of features such that no proper superset has the same support [HK00]. Frequent closed itemsets retain more information than their frequent maximal itemset counterparts, since frequent subsets of features are retained.

In Figure 3.4a, the closed frequent itemsets with a minimum support of 1 are mined from the clock sample set using the LCM algorithm by Uno et al. [UKA04] Only the closed frequent itemsets are shown, however, LCM is also capable of mining for maximal frequent itemsets.

The second step in constructing conjunctive association rules is the actual generation of rules using the mined frequent itemsets. We have implemented the association rule mining algorithm described by Agrawal et al. [AIS93] for this task. Using closed frequent itemsets, the mining procedure is as follows. Each closed frequent itemset I, is partitioned into two subset of features, $A \subset I$ and $B = I \setminus A$. An association rule, $\bigwedge_{x \in A} \Rightarrow \bigwedge_{y \in B}$ is constructed for each possible subset of features A. This process is repeated for each frequent itemset that is mined. Association rules that satisfy a certain user-specified minimum support and confidence thresholds, so called *strong association rules*, are kept

In Figure 3.4b, the conjunctive association rules mined from the clock sample set of the form $x \Rightarrow \bigvee Y$, where x is a single feature, and Y is a set of features, are shown. Conjunctive association rules with more than one feature in the antecedent have been omitted, since they do not have an effect on the feature hierarchy. Figure 3.4c is a listing of all the binary implications equivalent to the conjunctive association rules in Figure 3.4b. Note that the binary association rules with 100% confidence are equivalent to the binary implications, as discussed in Section 3.3.

Figure 3.5a shows the implication graph constructed from the mined binary association rules with 100% confidence (shown in Figure 3.4). In Figure 3.5b, the feature diagram constructed from the implication graph is shown. The bi-implications between Analog and Hands is translated into an AND-group. All other features are represented as optional features in the feature hierarchy.

3.4.2 Group Implications

Group implications have the form $f \rightarrow f_1 \vee \cdots \vee f_k$ where f is a feature group, and $f_1, \ldots, f_k$ are the set of subfeatures of the group. Group implications add additional constraints to a set of subfeatures and require that child-parent implications exist for its subfeatures. Thus, the implications necessary for constructing an OR-group f, with the set of subfeatures G are:

$$\begin{array}{lll} 1. \text{ child-parent implications:} & f_i \rightarrow f & \text{for all } f_i \in G \\ 2. \text{ group implication:} & f \rightarrow \bigvee\limits_{f_i \in G} f_i & \end{array} \tag{3.15}$$

1. Clock
2. Clock $\wedge$ Alarm
3. Clock $\wedge$ Analog $\wedge$ Hands
4. Clock $\wedge$ Alarm $\wedge$ Analog $\wedge$ Hands
5. Clock $\wedge$ Digital
6. Clock $\wedge$ Alarm $\wedge$ Digital
7. Clock $\wedge$ Digital $\wedge$ 24hr

(a) Closed frequent itemsets with minimum support of 1

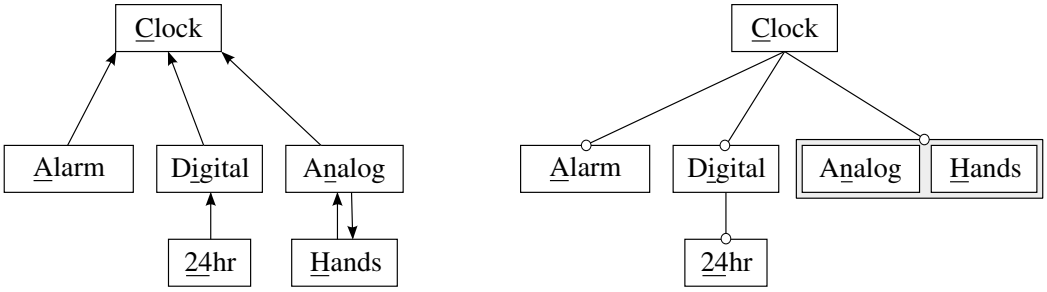
$$
\begin{aligned}
\text{Alarm} &\Rightarrow \text{Clock} && conf = 1.0\\
\text{Analog} &\Rightarrow \text{Clock} \wedge \text{Hands} && conf = 1.0\\
\text{Hands} &\Rightarrow \text{Clock} \wedge \text{Analog} && conf = 1.0\\
\text{Digital} &\Rightarrow \text{Clock} && conf = 1.0\\
\text{24hr} &\Rightarrow \text{Clock} \wedge \text{Digital} && conf = 1.0
\end{aligned}
$$

(b) Subset of the conjunctive association rules with 100% confidence

$$
\begin{aligned}
\text{Alarm} &\rightarrow \text{Clock}\\
\text{Digital} &\rightarrow \text{Clock}\\
\text{24hr} &\rightarrow \text{Clock}\\
\text{Analog} &\rightarrow \text{Clock}\\
\text{Hands} &\rightarrow \text{Clock}
\end{aligned}
\qquad
\begin{aligned}
\text{24hr} &\rightarrow \text{Digital}\\
\text{Analog} &\rightarrow \text{Hands}\\
\text{Hands} &\rightarrow \text{Analog}
\end{aligned}
$$

(c) Mined binary implications

Figure 3.4: Binary association rule mining



(a) Implication graph (b) Feature diagram

Figure 3.5: Clock implication graph and resulting feature diagram

Group implications are found by mining for disjunctive association rules in the sample set. In general, a *disjunctive association rule* is an association rule between two sets of features, X and Y, with the form [NCJK01]:

$$\bigwedge_{x \in X} x \Longrightarrow \bigvee_{y \in Y} y \tag{3.16}$$

In the context of feature model, we can restrict the mining to rules with a single feature in the antecedent such that the form of the rule is, $f \rightarrow f_1 \vee \cdots \vee f_k$. We call these rules *simple disjunctive association rules*.

Consequently, to retrieve the implications necessary to construct an OR-group (shown in Equation 3.15), the following forms of association rules with 100% confidence are mined:

$$\begin{array}{lll} 1. \text{ binary association rules:} & f_i \Rightarrow f & \text{for all } f_i \in G \\ 2. \text{ simple disjunctive association rule:} & f \Rightarrow \bigvee_{f_i \in G} f_i & \end{array} \tag{3.17}$$

The mining algorithm uses a novel approach for mining disjunctive association rules. First, the set of minimal OR-clauses are mined from the sample set. Afterwards, the disjunctive association rules are constructed using the minimal OR-clauses as the consequents.

An OR-clause is a disjunction of features where it has $\vee$ as the only operator over its literals, or features in our context [ZZR06]. The supporting configurations of an OR-clause, o, is the set of configurations that contain at least one feature in o. The *closure* of an OR-clause o, called $\mathbb{C}(o)$, is the union of all features contained in the supporting configurations, $\mathbf{c}(o)$. In other words, the closure of an OR-clause contains only the selected features in the configuration expression, $\mathbf{r}(\mathbf{c}(o))$.

A *minimal OR-clause* is defined as a disjunction of features such that no subset of features has the same closure (of features). The set of all minimal OR-clauses for a sample

Tidset	Minimal OR-clauses	
1	24hr	
1, 2	Digital	
2, 4	Alarm	
3, 4	Analog	Hands
1, 2, 4	Digital ∨ Alarm	24hr ∨ Alarm
1, 3, 4	24hr ∨ Analog	24hr ∨ Hands
2, 3, 4	Alarm ∨ Analog	Alarm ∨ Hands
1, 2, 3, 4	Clock Digital ∨ Hands 24hr ∨ Alarm ∨ Hands	Digital ∨ Analog 24hr ∨ Alarm ∨ Analog

Table 3.2: Minimal OR-clauses mined from the clock sample set

set, $\mathcal{M}$, is defined as the set of minimal OR-clauses for all combinations of closures over the sample set. The definition of $\mathcal{M}$, where O and X are sets of features, and $\mathbb{C}$ is the closure operator described above is:

$$\mathcal{M} \leftarrow \left\{ \bigvee O \;\middle|\; \nexists X \subset O \text{ s.t. } \mathbb{C}\left(\bigvee X\right) \equiv \mathbb{C}\left(\bigvee O\right) \right\} \tag{3.18}$$

Intuitively, a minimal OR-clause can be understood as the smallest expression that forms a lossless representation of all possible OR-clauses [ZZR06]. Using the closure operator on a minimal OR-clause o, all other selected features in the set of supporting configurations can be retrieved. The mined minimal OR-clauses from the clock sample set are shown in Table 3.2.

Recall the intent was to use minimal OR-clauses as a means of constructing simple disjunctive association rules with the form $f \Rightarrow f_1 \vee \cdots \vee f_k$. A minimal OR-clause describes a disjunction amongst the subfeatures of a group. Thus, the minimal OR-clause is used as the consequent of a disjunctive association rule. Features present in one or more configurations that the minimal OR-clause satisfies, or collocated, are selected as the antecedent. A feature f is *collocated* with a minimal OR-clause o iff the supporting configurations of o, $\mathbf{c}(o)$ is a subset of the supporting configurations of f, $\mathbf{c}(f)$.

The set of disjunctive association rules, $\mathcal{D}$, is mined using the conditions shown in Equation 3.19. In the case where the set of supporting configurations of the antecedent, $\mathbf{c}(f)$, is equal to the set of supporting configurations of the minimal OR-clause, $\mathbf{c}(o)$, the resulting association rule will have 100% confidence.

$$\mathcal{D} \leftarrow \{ f \Rightarrow o \mid o \in \mathcal{M} \;\wedge\; \mathbf{c}(o) \subseteq \mathbf{c}(f) \} \tag{3.19}$$

The simple disjunctive association rules constructed from the mined minimal OR-clauses (in Table 3.2) with 100% confidence are shown in Table 3.3. These simple disjunctive

$24hr \rightarrow Digital \vee Alarm$	$Digital \rightarrow 24hr \vee Alarm$
Clock $\rightarrow$ Digital $\vee$ Analog	**Clock $\rightarrow$ Digital $\vee$ Hands**
$24hr \rightarrow Digital \vee Analog$	$24hr \rightarrow Digital \vee Hands$
$Alarm \rightarrow Digital \vee Analog$	$Alarm \rightarrow Digital \vee Hands$
$Hands \rightarrow Digital \vee Analog$	$Hands \rightarrow Digital \vee Hands$
$Hands \rightarrow 24hr \vee Analog$	$Hands \rightarrow Alarm \vee Analog$
$Analog \rightarrow 24hr \vee Hands$	$Analog \rightarrow Alarm \vee Hands$
Clock $\rightarrow$ 24hr $\vee$ Alarm $\vee$ Analog	**Clock $\rightarrow$ 24hr $\vee$ Alarm $\vee$ Hands**
$Digital \rightarrow 24hr \vee Alarm \vee Analog$	$Digital \rightarrow 24hr \vee Alarm \vee Hands$
$Hands \rightarrow 24hr \vee Alarm \vee Analog$	$Analog \rightarrow 24hr \vee Alarm \vee Hands$

Table 3.3: Mined group implications

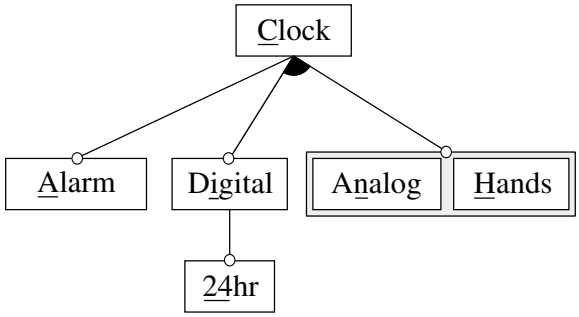
rules are written as their equivalent implication. The implications shown in bold face are implications of the correct form where the feature in the antecedent is an ancestor of the subfeatures in the consequent. Thus, only these rules shown in bold have an effect on the feature hierarchy.

In Figure 3.6, the feature diagram retrieved after adding the mined group implications to the feature hierarchy formula is shown. Two OR-groups are created as a result of the new implications. First, an OR-group between Digital and the AND-group (Analog $\wedge$ Hands) is shown in the hierarchy. The second OR-group between Alarm or 24hr or (Analog $\wedge$ Hands) is shown as an additional constraint since it cuts through the feature hierarchy (ie. Clock is not a direct parent of 24hr).

Disjunctive association rule mining is not as well researched as that of conjunctive association rule mining. Nanavati et al. [NCJK01] present a method for finding disjunctive association rules through a greedy search algorithm. However, their method does not retrieve the complete set of disjunctive association rules for a given sample set and as a result, is not applicable for mining feature models.

3.4.3 Mutual Exclusion Clauses

A *mutual exclusion clause* has the form $f_i \rightarrow \overline{f_j}$ where f_i and f_j are subfeatures of an OR-group. If mutual exclusion clauses exist between all subfeatures of an OR-group, then a XOR-group is constructed in its place. The conditions necessary for constructing a



$$\text{Clock} \rightarrow \text{Alarm} \vee \text{24hr} \vee (\text{Analog} \wedge \text{Hands})$$

Figure 3.6: Feature diagram after adding group implications

XOR-group f with the set of subfeatures G are:

$$\begin{array}{lllll} 1. & \text{child-parent implications:} & f_i \rightarrow f & \text{for all } f_i \in G & \\ 2. & \text{group implication:} & f \rightarrow \bigvee_{g \in G} g & & (3.20) \\ 3. & \text{mutual exclusion clauses:} & f_i \rightarrow \overline{f_j} & \text{for all } f_i, f_j \in G \text{ and } i \neq j & \end{array}$$

Mutual exclusion clauses are equivalent to so called *negative binary association rules* with 100% confidence of the form:

$$f_i \Rightarrow \overline{f_j} \tag{3.21}$$

Thus, the following forms of association rules with 100% confidence are mined in order to find the implications necessary to construct an XOR-group f with subfeatures G:

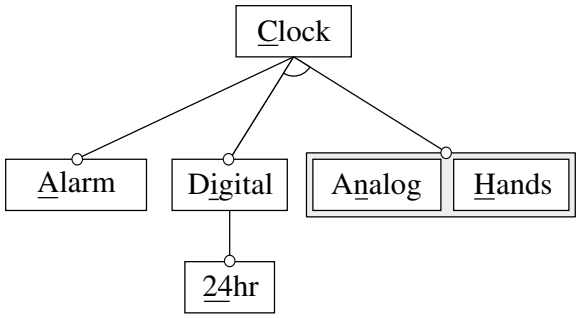
$$\begin{array}{lllll} 1. & \text{binary association rules:} & f_i \Rightarrow f & \text{for all } f_i \in G & \\ 2. & \text{simple disjunctive association rule:} & f \Rightarrow \bigvee_{f_i \in G} f_i & & (3.22) \\ 3. & \text{negative binary association rules:} & f_i \Rightarrow \overline{f_j} & \text{for all } f_i, f_j \in G \text{ and } i \neq j & \end{array}$$

A significant difference with mining for negative binary association rules compared with other expression forms is the need to mine on the absent features of a configuration, $\overline{c}$. Similar to mining for binary association rules, two approaches have been explored to retrieve negative binary association rules: (i) mining binary implications on absent features in the sample set, or (ii) using conjunctive association rule mining with absent features.

Feature combinations with absent features. This approach is similar to the mining procedure for binary association rules in Section 3.4.1 where we iterate through all feature

Digital $\rightarrow$ **$\overline{\text{Analog}}$**	24hr $\rightarrow \overline{\text{Alarm}}$
Digital $\rightarrow$ **$\overline{\text{Hands}}$**	24hr $\rightarrow \overline{\text{Analog}}$
Digital $\rightarrow \overline{\text{24hr}}$	24hr $\rightarrow \overline{\text{Hands}}$

Table 3.4: Mined mutual exclusion clauses



$$\text{Clock} \rightarrow \text{Alarm} \vee \text{24hr} \vee (\text{Analog} \wedge \text{Hands})$$

Figure 3.7: Feature diagram after adding mutual exclusion clauses

combinations. The key difference is that the consequent of the rules consists of absent features. We define a new set, $\overline{\mathcal{F}}$, which contains the set of all negated features $\overline{f}$, such that $f \in \mathcal{F}$. Using this approach, a negative binary association rule $x \Rightarrow \overline{y}$ is constructed for all feature tuples $(x, \overline{y}) \in \mathcal{F} \times \overline{\mathcal{F}}$.

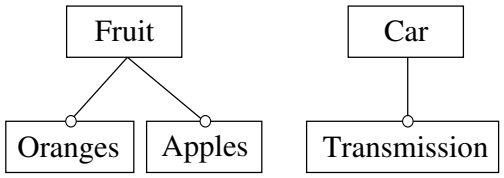
Conjunctive association rule mining with absent features. Conjunctive association rule mining can also be used to retrieve negative binary association rules. The employed association rule mining algorithm by Agrawal et al. [AIS93] is capable of mining only on selected features. Thus, we apply an operator $\mathbf{e}(\mathcal{S})$ to the sample set, which appends additional features, f^{-}, that represent absent features in a configuration:

$$\mathbf{e}(\mathcal{S}) = \{c \cup \{f^{-} \mid f \in \overline{c}\} \mid c \in \mathcal{S}\} \tag{3.23}$$

In Table 3.4, the mined negative binary rules with 100% confidence are shown. These rules are presented as their equivalent mutual exclusion clauses. The mutual exclusion clauses in bold face are the clauses that are applicable to the mined OR-groups. Adding these two mutual exclusion clauses to the feature hierarchy formula, the OR-group between *Digital* and *Analog* $\wedge$ *Hands* is further constrained into a XOR-group. The final mined feature model is shown in Figure 3.7.

tid	configuration
1	Fruit
2	Fruit Apples Oranges
3	Car
4	Car Transmission

(a) sample set



(b) Mined feature models

Figure 3.8: Retrieving multiple feature models

3.4.4 Retrieving Multiple Feature Models

The mining algorithm is capable of retrieving multiple feature models from a single sample set. This situation arises when there are two sets of unrelated configurations tangled in the sample set. For example, assume that the set of features, $\mathcal{F}$, was partitioned into two sets F_A and F_B such that $F_A \cap F_B = \mathcal{F}$. If there are two sets of configurations, C_A and $C_B \in \mathcal{S}$, such that C_A contains only features from F_A and C_B contains only features from F_B, then the mined association rules will have disjoint feature spaces. As a result, two feature models will be constructed by the Czarnecki and Wąsowski synthesis algorithm. One feature model will be representative of the configurations in C_A, while the other will represent the configurations in C_B.

Figure 3.8 illustrates this scenario. In the sample set, four configurations are present. Configurations 1 and 2 correspond to fruit, while configurations 3 and 4 correspond to a car. As a result, two separate feature models are retrieved from the sample set.

3.4.5 Additional Hard Constraints

Hard constraints in a probabilistic feature model can be separated into two categories: (i) feature hierarchy, and (ii) additional hard constraints. The set of additional hard constraints appear as a list of propositional formulas below the feature hierarchy. Additional hard constraints can be used form constraints that would otherwise complicate the feature hierarchy.

Currently, only group implications which cut across the feature hierarchy are considered additional hard constraints. These group implications are have subfeatures which are not direct children of the parent feature group.

For example, in the clock example shown in Figure 3.7 on the preceding page, the group constraint, $Clock \rightarrow Alarm \vee 24hr \vee (Analog \wedge Hands)$, is shown as an additional hard constraint because 24hr is not a direct child of Clock.

Clock $\Rightarrow$ Digital $\vee$ Alarm	$supp = 3$	$conf = 0.75$
Clock $\Rightarrow$ 24hr $\vee$ Alarm	$supp = 3$	$conf = 0.75$
Clock $\Rightarrow$ 24hr $\vee$ Analog	$supp = 3$	$conf = 0.75$
Clock $\Rightarrow$ 24hr $\vee$ Hands	$supp = 3$	$conf = 0.75$
Clock $\Rightarrow$ Alarm $\vee$ Analog	$supp = 3$	$conf = 0.75$
Clock $\Rightarrow$ Alarm $\vee$ Hands	$supp = 3$	$conf = 0.75$

Table 3.5: Set of soft constraints for the clock feature model

3.4.6 Soft Constraints

The last component to retrieve for a probabilistic feature model are its soft constraints. The set of soft constraints describes relationships among the features which exist in most sample configurations, but not all. Soft constraints are useful for describing suggested defaults or trends.

The mining algorithm filters association rules with confidence greater than 50% as soft constraints. A confidence of 50% for an association rule $A \Rightarrow B$ implies that the antecedent A has no effect on the consequent B. Thus, a confidence of greater than 50% implies a positive relationship where the presence of the antecedent A increases the chance of the consequent B appearing.

In Table 3.5, the soft constraints for the clock example are shown. In the left column, the soft constraint is shown with the support and confidence measures to the right. Only disjunctive association rules are present in this example, however in general, any mined rule may be a soft constraint if the confidence is greater than 50% and less than 100%. In practice, the minimum confidence threshold is often set higher than 50% in order to limit the mined soft constraints to a manageable number.

3.5 Propositional Formula to Feature Model

A *feature hierarchy formula* is constructed from mined association rules with 100% confidence. The feature hierarchy formula is used as the input propositional formula to the feature model synthesis algorithm [CW07].

The feature model synthesis algorithm constructs a feature diagram using a propositional formula as input. The feature hierarchy represent only hard constraints which must be satisfied in all legal configurations. As a result, only rules with 100% confidence are added to the feature hierarchy formula. Rules with 100% confidence are satisfied for the entire sample set, and thus, are equivalent to a Boolean implication.

The algorithm separates the different forms of formulas for a basic feature model given the feature hierarchy formula. First, an implication hypergraph is constructed from the feature hierarchy formula. An implication hypergraph is a visualization of the feature hierarchy formula where the nodes correspond to features and constants (*true* or $false$), and the hyperedges correspond to implications. The hyperedges can have multiple sources and targets. Each hyperedge represent a conjunction of source features which imply a disjunction of the target features.

A summary of the procedure is as follows. The first step in the algorithm is identifying feature hierarchy implications in the feature hierarchy formula. Next, AND-groups are identified in the implication hypergraph as cliques. The AND-groups are constructed, and the cliques are then removed. Finally, OR-groups and XOR-groups are constructed. The feature model synthesis algorithm is implemented using operations on binary decision diagrams (BDDs) [Bry86].

When retrieving a feature model from a propositional formula, it is possible for a feature to have multiple parents. For example, two implications $B \rightarrow A$ and $C \rightarrow A$ in the propositional formula will result in the feature A having two parents, B and C. Since it is impossible to selective choose a single parent without additional information, the resulting synthesized feature model is in a generalized feature model notation [CW07], that allows features to have multiple parents.

3.6 Feature Model Mining Algorithm

In Figure 3.9, the feature model mining algorithm is presented. Two forms of association rule mining are used: (i) conjunctive, and (ii) disjunctive association rule mining. Four parameters are provided as input to the algorithm: the sample set $\mathcal{S}$, the minimum support threshold $\check{s}$, the maximum support threshold $\hat{s}$, and the minimum confidence m. The $conf(\alpha, \mathcal{S})$ and $supp(\alpha, \mathcal{S})$ operators returns the confidence and support of the given association rule α, with respect to the sample set $\mathcal{S}$. Association rules which satisfy the minimum support and minimum confidence are called *strong association rules*.

The algorithm begins by mining conjunctive association rules in lines 3–4 using the algorithm described by Agrawal et al. [AIS93]. The conjunctive association rules are used to retrieve binary implications and mutual exclusion clauses. Mining on the set of selected features retrieves binary association rules. However, in order to retrieve negative binary association rules, the set of absent features must be included for each sample configuration. The conjunctive rule mining algorithm is capable of mining only on selected features, thus, the $\mathbf{e}(\mathcal{S})$ operator, defined in Equation 3.23 on page 25, is used to append additional features to each configuration in order to represent absent features.

Next, minimum OR-clauses are mined in lines 5–6 using the BLOSOM-MO algorithm by Zhao et al. [ZZR06]. These OR-clauses are a prerequisite for constructing disjunctive association rules. The maximum support is used to limit the number of minimal OR-clauses that are mined since an OR-clause with excessive support is likely too general to be useful as a group implication.

MINE-FEATURE-MODEL($\mathcal{S}$: sample set; $\check{s}, \hat{s}, m : [0,1]$)

1 ▷ $\check{s}$ is min. support and $\hat{s}$ is max. support
2 ▷ m is min. confidence

3 ▷ Mine strong conjunctive association rules [AIS93]
4 C $\leftarrow$ CONJ-RULE-MINER($\mathbf{e}(\mathcal{S}), \check{s}, m$)

5 ▷ Mine frequent minimal OR-clauses [ZZR06]
6 $O \leftarrow$ BLOSOM-MO($\mathcal{S}, \check{s}, \hat{s}$)

7 ▷ Build disjunctive association rules
8 D $\leftarrow \{d \leftarrow p \Rightarrow f_1 \vee \cdots \vee f_k \mid$
9 $\quad$ supp(d,$\mathcal{S}$) $\geq \check{s} \wedge conf(d, \mathcal{S}) \geq c$
10 $\quad \wedge f_1 \vee \cdots \vee f_k \in O$
11 $\quad \wedge \mathbf{c}(f_1 \vee \cdots \vee f_k) \subseteq \mathbf{c}(p) \}$

12 ▷ Select hard constraints
13 H $\leftarrow \{b \leftarrow (f_i \Rightarrow f_j \in C) \mid conf(b, \mathcal{S}) = 1\}$
14 $\quad \cup \{g \leftarrow (p \Rightarrow f_1 \vee \cdots \vee f_k) \in D \mid conf(g, \mathcal{S}) = 1\}$
15 $\quad \cup \{e \leftarrow (f_i \Rightarrow \overline{f_j} \in C) \mid conf(e, \mathcal{S}) = 1\}$

16 ▷ Create set of soft constraints
17 $S \leftarrow (B \cup G \cup M) \setminus H$

18 ▷ Synthesize feature model [CW07]
19 $G \leftarrow$ FEATURE-GRAPH($\bigwedge_{r \in H} r$)

20 ▷ Return mined feature model and soft constraints
21 **return** G, S

Figure 3.9: Feature model mining algorithm [CSW08]

The set of simple disjunctive association rules are mined in lines 7–11. We call these rules *simple* to distinguish the fact that only a single feature exist in the antecedent. These rules are constructed by using the minimal OR-clauses as the consequent and selecting features that appears in the same supporting set of configurations as its antecedent.

The hard constraints for constructing the feature hierarchy are selected in lines 12–15 of the algorithm. Hard constraints are association rules with the correct form and have a confidence equal to 100%. The set b contains the association rules equivalent to binary implications, g contains rules equivalent to group implications and finally, e contains the rules equivalent to mutual exclusion clauses.

The set of soft constraints is constructed in lines 16–17. The soft constraints are the remaining rules after the hard constraints have been removed. All soft constraints have less than 100% confidence.

Finally, in line 18, the propositional formulas are conjoined to form the feature hierarchy formula and the feature model synthesis algorithm [CW07] is executed.

The feature model mining algorithm returns the mined feature hierarchy and a set of soft constraints.

3.7 Implementation

The feature model mining algorithm has been implemented as a prototype Eclipse IDE plug-in [Ecl08a]. A prototype implementation of the feature model synthesis algorithm was graciously provided by Andrzej Wąsowski. The feature diagrams are generated using the GraphViz graph visualization project [ATT08] or visualized in Eclipse using the Zest toolkit [BBS04].

Frequent itemsets are mined using the LCM miner by Uno et al. [UKA04]. Conjunctive association rules are constructed using the procedure described by Agrawal et al. [AIS93]. Minimal OR-clauses are mined using the BLOSOM-MO algorithm by Zhao et al. [ZZR06]. A language describing for sample sets and configurations was constructed using the openArchitectureWare xText textual DSL generator [ope08].

A source code analyzer for extracting basic code patterns was constructed using the Eclipse Java Development Tools (JDT) project [Ecl08b]. The Applet FSML analyzer wizard [AC07] is used to construct the applet framework-specific models described in the next chapter.

CHAPTER 4

Mining on Frameworks

Object-oriented frameworks are a set of reusable abstractions commonly used for building software systems. A framework models a domain and have a set of rules that developers must follow. A *framework-specific modeling language (FSML)* is a domain-specific language that models concepts and the relations in an object-oriented framework [AC06]. In this chapter, the feature model mining algorithm is applied towards the development and maintenance of object-oriented frameworks and FSMLs.

An FSML consists of a metamodel, which formalizes the framework-provided concepts and relations and optionally, an interpreter, which constructs a mapping between the FSML metamodel and code. The FSML metamodel is represented as an extended feature model with support for feature cardinalities, cloning, references and attributes. The semantics of the features are specified using *mapping definitions* [ABC07]. The mapping definitions are implemented for a particular language, such as Java, using so called *mapping interpreters* [Ant08].

An FSML is capable of reverse-engineering a framework-specific model by using its specified mapping definitions and a mapping interpreter. A *framework-specific model*, is a model of the framework-provided concepts as implemented by a sample application. A framework-specific model can be used as a comprehension artifact or as a means of evolving the application through forward-engineering [ABC07]. FSMLs provide a notation for model-driven engineering [Sch06] or the Object Management Group's variant, model driven architecture [Obj04]. A framework-specific model can also be specified a priori, allowing a user to specify a model in which framework completion code can be automatically generated.

Feature model mining can be used to retrieve a feature model representing code patterns in a set of sample applications. The mined feature model is useful for FSML development in the following three use cases: (i) jump-starting development of an FSML, (ii) confirmation or comparison of an FSML with existing framework applications and, (iii) refining an existing FSML by discovering unexpected patterns in framework applications. These use cases will be described in further detail below.

(i) *Jump-starting FSML development*. A mined feature model with features representing structural and behavioral patterns will provide a model of the variation points in the sample applications of a framework. The variation points in the feature model are potential framework hotspots or areas requiring developer customization.

(ii) *Comparison of an FSML and framework applications*. Feature model mining can be applied to compare an existing expert-specified FSML with the framework usages present in a set of sample applications.

(iii) *Refining an existing FSML by discovering unexpected patterns in framework usage*. Feature model mining can be used to discover common implementation patterns among the sample applications themselves. The mined feature model represents the framework usage patterns in the sample applications and can be used as a framework comprehension artifact.

In this chapter, the feature model mining algorithm is applied to a set of configurations constructed using the Applet FSML. The expert-specified Applet FSML is first described in Section 4.1.

In Section 4.2, the Applet FSML will be reverse-engineered using the feature model mining algorithm on a sample set of applets. This approach assumes that a mapping interpreter and a set of code queries are defined prior to the mining approach. Consequently, the process described in this section addresses use cases (ii) and (iii).

In Section 4.3, a brief discussion on a methodology for mining in the scenario where a mapping between framework-provided features and code does not exist. We propose mining on the framework boundary. This approach address use case (i), where the resulting mined feature model can be used as a means of jump-starting FSML development.

4.1 Applet FSML

The Applet FSML models the concepts and suggested usages of the Java Applet framework [AC07]. An applet is a Java program that can be included in a web page and run from a web browser [Sun08]. The expert-specified metamodel of the Applet FSML is shown in Figure 4.1. A feature with an exclamation mark is an *essential* feature. Essential features are a stronger form of constraint than mandatory features and *must* be present if it's parent feature is present. The absence of an essential feature implies that it's parent feature is also absent. The absence of a mandatory features, on the other hand, simply indicates an error in the configuration. The remainder of this section will describe the framework-provided concepts in the model.

The root feature in the Applet FSML metamodel is the AppletModel, which contains a list of applets.

The name feature represents the fully-qualified Java class name for an applet. extendsApplet is selected in a configuration if the Applet class extends `java.applet.Applet`. The extendsJApplet feature corresponds to whether the applet extends `javax.swing.JApplet` (which is a subclass of `java.applet.Applet`).

Applet
- [1..1] name (String)
- **!**[1..1] extendsApplet
  - [0..1] extendsJApplet
- [0..∗] parameter
  - [0..∗] name
- [0..1] providesParameterInfo
- [0..∗] showsStatus
  - [0..∗] message (String)
- [1..1] overridesRequiredMethods
  - ⟨**1**–**3**⟩
  - [0..1] overridesInit
  - [0..1] overridesStart
  - [0..1] overridesPaint
- [0..∗] registersMouseListener
  - ⟨**1**–**1**⟩
  - [0..1] this
    - [1..1] implementsMouseListener
    - [1..1] deregisters
      - **!**[1..1] this
  - [0..1] mouseListenerField
    - [1..1] fieldName (String)
    - [1..1] typedMouseListener
    - [1..1] deregisters
      - **!**[1..1] this

Applet (cont.)
- [0..∗] thread
  - **!**[1..1] typedThread
  - [1..1] nullifiesThread
  - [1..1] initializesThread
    - ⟨**1**–**1**⟩
    - [0..1] initializesThreadWithRunnable
      - ⟨**1**–**1**⟩
      - [0..1] this
        - [1..1] implementsRunnable
      - [0..1] helper
      - [0..1] variable
      - [0..1] runnableField
        - [1..1] typedRunnable
        - [1..1] name (String)
    - [0..1] initializesThreadWithSubclass
      - [1..1] name (String)
      - [1..1] overridesRun

Figure 4.1: Expert-specified Applet FSML metamodel

The overridesRequiredMethods OR-group models the rule that an Applet should override one or more lifecycle methods: `init`, `start` and `paint`. These methods are shown as the grouped subfeatures: overridesInit, overridesStart, overridesPaint.

The features showsStatus and message represent the calls and the strings passed as parameters to the method `Applet.showsStatus(String)` which displays a message in the status bar of the applet window.

The XOR-group registersMouseListener, model the registration of `MouseListeners` through calls to `java.awt.Component.addMouseListener(...)`. Two subfeatures exist under the listener groups, this and mouseListenerField. The feature, this, represents the applet object was provided as a parameter to registersMouseListener call and mouseListenerField models the case where the parameter was a field.

The implementsMouseListener mandatory feature represents the rule that an applet must implement a `MouseListener` if it is used as the parameter to the register call. The deregisters / this mandatory features model the framework requirement that `java.awt.Component.removeMouseListener(...)` must also be invoked with the applet object (`this`) as a parameter.

Applet documentation recommends long running operations to be executed as background threads. The next feature, thread, models the threads and the various methods of instantiation. The essential feature, typedThread, indicates that the thread must be of type `java.lang.Thread`. The initializesThread XOR-group represent the two alternative methods of instantiating a thread. In the first method, the thread could be initialized with this, where the applet implements `java.lang.Runnable`. Otherwise, the thread must be initialized with a Thread subclass, which is shown by the initializesWithThreadSubclass feature.

Finally, the last group of features in the Applet FSML represent HTML parameters. Applets are intended to be embedded within web pages, where parameters are passed as HTML elements. parameter models accesses through the `Applet.getParameter(...)` method. Framework documentation also recommends applets provide parameter information by overriding the `getParameterInfo()` method which is represented by the providesParameterInfo feature.

For further details on the Applet FSML, refer to the technical report by Antkiewicz and Czarnecki [AC07].

4.2 Constructing the Applet Metamodel

We address use case (ii), where an existing FSML is compared to framework applications by using feature model mining on a set of applets to reconstruct an Applet FSML. The goal of the experiment was to evaluate the effectiveness of the mining algorithm on a sample set representing applications used in practice. The results of the analysis can be used to refine an existing FSML with additional tool support. This application of feature model mining is potential future work and is discussed further in Section 5.4.3.

The experiment used a sample set that was comprised of 63 applets gathered from tutorials by Sun Microsystems and various sources over the internet. The full list of applets used to construct the sample set is shown in Appendix A. Three applets from the original sample set provided by Michał Antkiewicz, *Hitmeter*, *MyApplet*, and *Ungrateful*, did not override one of the three required lifecycle methods (ie. `init`, `start`, `stop`) and were manually removed from the sample set prior to mining.

This section is structured as follows. In Section 4.2.1, the procedure of constructing a valid sample configuration from the applet source code is described. In Section 4.2.2, the mined applet feature model is presented. In Section 4.2.3, the soft constraints gathered from the mining process are presented and analyzed.

4.2.1 Constructing the Sample Set

A sample set must be constructed from the applet source code prior to the execution of the mining algorithm. We use the reverse-engineering capabilities of the Applet FSML to accomplish this task. The mapping definitions in the FSML map framework-provided concepts in the model to code patterns in implementation code.

Code patterns are classified into two categories: (i) structural or (ii) behavioral patterns [ABC07]. A structural pattern represents the static properties present in code. In a Java class, structural patterns include its inheritance hierarchy, implemented interfaces, fields, and methods. The second category, behavioral patterns, describe run-time properties, such as method calls, variable assignments and temporal properties such as the order of execution for a set of methods. A *code query* retrieves code patterns from an application implementation using static analysis. Code queries retrieve exact structural patterns and approximate behavioral patterns. The various code queries are not described in detail in this thesis, for further details, refer to relevant paper by Antkiewicz et al. [ABC07].

A reverse engineering algorithm by Antkiewicz constructs a framework-specific model for a given applet using the mapping definitions in the Applet FSML metamodel [Ant08]. The set of construct framework-specific models correspond to the sample set used for feature model mining. The framework-specific models use feature modeling extensions such as cardinalities, feature types and references which are unsupported in the current mining algorithm. As a result, the framework-specific models must be further processed before the configuration is suitable for our mining algorithm.

A basic feature can either be selected or absent (unselected) from a configuration. A cloneable feature, on the other hand, can have multiple copies depending on its specified cardinality. A cloneable feature in a configuration, c, is converted to a *selected feature* if at least one copy of the feature exists in c. A String feature is considered selected if the string is not `null`. Boolean features are translated such that *true* corresponds to the feature being selected. In addition, feature model references [CK05] are unfolded.

To demonstrate how a sample configuration is constructed, a simple Java applet is shown in Figure 4.2a. The FSML reverse engineering algorithm is executed on the applet

```
public class MyApplet extends java.applet.Applet {
    @Override
    public void init() {
        showStatus("Hello World!");
        String hw = "Hello World, again!";
        showStatus(hw);
    }
}
```

(a) A simple Java applet

- Applet
  - name “MyApplet”
  - extendsApplet
  - overridesRequiredMethods
    - overridesInit
  - showsStatus
    - message “`Hello World!`”
  - showsStatus
    - message “`Hello World, again!`”

(b) Framework-specific model

- Applet
- name
- extendsApplet
- overridesRequiredMethods
- overridesInit
- showsStatus
- message

(c) Sample configuration

Figure 4.2: Constructing a configuration from a framework-specific model

```
1: Applet ∧ name ∧ extendsApplet ∧ overridesRequiredMethods
2:   ⟨1–3⟩
3:     [0..1] overridesStart
4:       [0..1] initializesThreadWithSubclass ∧ initializesThreadWithSubclass.name
                ∧ initializesThreadWithSubclass.overridesRun ❶
5:     [0..1] overridesPaint
6:       [0..1] MouseListener.deregisters ∧ MouseListener.deregistersThis ❷
7:     [0..1] overridesInit
8:       [0..1] thread ∧ typedThread
9:         [0..1] initializesThread
10:          [0..1] nullifiesThread
11:          ⟨1–1⟩
12:            [0..1] implementsRunnable ∧ initializesThreadWithRunnable
                      ∧ initializesThreadWithRunnable.this
13:            [0..1] initializesThreadWithSubclass ∧ initializesThreadWithSubclass.name
                      ∧ initializesThreadWithSubclass.overridesRun ❶
14:      [0..1] registersMouseListener ∧ registersMouseListener.this ∧ implementsMouseListener
15:        [0..1] extendsJApplet
16:        [0..1] deregisters ∧ deregisters.this ❷
17:      [0..1] parameter ∧ name
18:        [0..1] providesParameterInfo
19:        [0..1] showsStatus
20:          [0..1] message
```

Figure 4.3: Mined Applet FSML metamodel

code, and the resulting framework-specific model is shown in Figure 4.2b. The framework-specific model is converted into a suitable configuration for data mining as shown in Figure 4.2c.

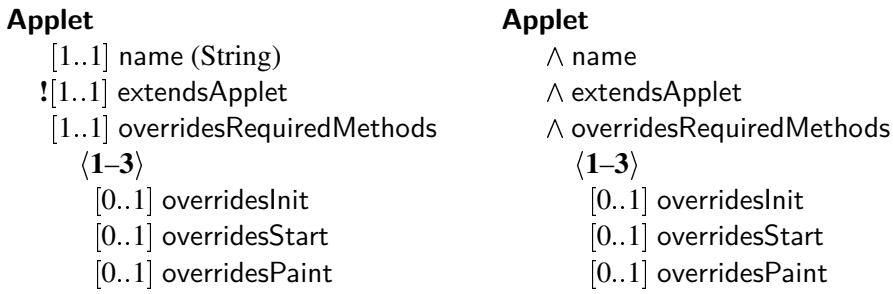
4.2.2 Mined Applet Metamodel

After the sample set is constructed, the mining algorithm is executed. The resulting mined feature model is shown in Figure 4.3. Note that the mined Applet metamodel is in the generalized feature model notation as described in Section 3.5, which allow features to have multiple parents. Features marked with ❶ or ❷ both have two parents and appear in the textual feature model twice. In addition, AND-groups are denoted with the ∧ symbol between its features. The mined applet metamodel will be discussed in several groups of features.

Overall. For the most part, related features remained together in several trees in the mined model. A significant difference between the expert-specified model and the mined

model lies in the feature cardinalities, references and attributes and lack of mandatory features. The mining algorithm described in this thesis is unable to retrieve $[0..*]$ cardinalities or feature references. All features are reduced to basic features which can be selected or unselected prior to mining. Thus, features are either optional, or part of an AND-group. AND-groups can be separated into a hierarchy of mandatory features, however, additional knowledge is required. This extension to the mining algorithm is discussed further in Section 5.4.1. In addition, essential features are absent from the mined model. Additional knowledge from a framework expert is necessary to separate mandatory and essential features. The expert could add essential constraints to features through a feature model editor such as the Feature Model Plug-in, *fmp* [CAK+05].

Applet root. The mined feature model is quite similar to the structure of the expert-specified model in this set of features. The mandatory features name and extendsApplet are preserved in the mined feature model as an AND-group. The OR-group nested under overridesRequiredMethods is preserved in the mined feature model, however, the feature itself is collapsed as part of an AND-group.



(a) Expert (b) Mined

Parameter and showsStatus. An interesting result of the mining algorithm is present in this set of features. First, the parameter and showsStatus features are nested under the overridesInit feature. This set of features correspond to method calls, and consequently, must be nested within a method. In our sample set, the parameter and showsStatus all co-occur with an overridden `init()` method. Note that nesting relation in this case does not described that the calls for parameter and showsStatus are *within* the `init()` method, but rather describes a co-occurrence between the two features.

The providesParameterInfo and showsStatus features are subfeatures of parameter in the mined metamodel. In the case of providesParameterInfo, there is no framework rule that an applet must provide its own parameter information if parameters are accessed. However, examining applets in the sample set, a trend emerges where if parameter information is provided, then external parameters are also accessed. This pattern is reflected in the nesting relationship between the features in the mined feature model. Similarly with showsStatus, the nesting relationship with parameters is not specified in applet documentation, but is a common pattern exhibited in our sample set.

[0..∗] parameter
  [0..∗] name
[0..1] providesParameterInfo
[0..∗] showsStatus
  [0..∗] message (String)

(a) Expert

[0..1] parameter
  ∧ name
    [0..1] providesParameterInfo
    [0..1] showsStatus
      [0..1] message

(b) Mined

Threads. The typedThread feature is part of an AND-group with thread in the mined metamodel. This is an expected side effect of the mining algorithm as discussed earlier. The XOR-group nested under initializesThread is also preserved in the mined metamodel. Similar to the previous set of features, thread is a child of the overridesInit feature. This relation occurs because all applets with threads, also override the `init()` method.

The initializesThread and nullifiesThread features were both mandatory features in the expert-specified model, but in the mined model, both features appear as optional.

First, the change in the feature type for initializesThread was traced back to the *DotProduct* sample applet. In this applet, a thread field was declared, but was simply initializes to `null`. This field was not access in the applet at all. As a result, it's framework-specific model contained an instance of thread and typedThread, but not of initializesThread.

Similarly, with the nullifiesThread feature, the feature was retrieved as an optional feature due to an erroneous sample applet. In this case, the applet had declared and initialized the thread, but did not nullify the thread after its use.

The nesting relation between initializesThread and nullifiesThread is explained by the aforementioned erroneous applets. In one, a thread was declared but not initialized. In the other, the thread was declared and initialized, but not nullified. Thus, the mining algorithm constructed nullifiesThread as a child of initializesThread.

[0..∗] thread
  ![1..1] typedThread
  [1..1] nullifiesThread
  [1..1] initializesThread
    ⟨**1**–**1**⟩
      [0..1] withRunnable
        ...
      [0..1] withThreadSubclass
        ...

(a) Expert

[0..1] thread
  ∧ typedThread
  [0..1] initializesThread
    [0..1] nullifiesThread
    ⟨**1**–**1**⟩
      [0..1] withRunnable
        ...
      [0..1] withThreadSubclass
        ...

(b) Mined

Initializes thread with subclass. The mandatory features nested under InitializesWithThreadSubclass are appropriately preserved as an AND-group in the mined model.

The mining algorithm discovered an interesting pattern with the AND-group containing the features InitializesWithThreadSubclass, name and overridesRun. In the mined feature

model, the AND-group has two parents, overridesStart and overridesInit. The relationship with these methods is reasonable since threads need to be explicitly started in a method, and the `start` or `init` methods seems like an appropriate place to do so.

[0..1] initializesWithThreadSubclass
  [1..1] name (String)
  [1..1] overridesRun

(a) Expert

[0..1] initializesThreadWithSubclass
  ∧ name
  ∧ overridesRun

(b) Mined

Initializes thread with runnable. The initializesThreadWithRunnable feature is nested under the overridesInit feature, but not the overridesStart feature like the previous group of features. The initializesThreadWithRunnable feature models the scenario when the applet class is used as the thread. Since the thread does not need to be explicitly instantiated, it is likely that application developers started the thread when the applet is initialized. This pattern is exhibited in the sample set, and is reflected in the mined feature model.

The remaining features have significant differences between the mined feature model and the expert-specified metamodel. The helper, variable and runnableThread features are absent in the mined model. These features are missing due to the absence of these features in the sample applets themselves. The XOR-group under initializesThreadWithRunnable is also absent. The absence of the forementioned features caused the mined model to have no alternatives other than this. As a result, no group is constructed, and this (feature) is collapsed into the AND-group.

[0..1] initializesThreadWithRunnable
  ⟨**1–1**⟩
  [0..1] this
    [1..1] implementsRunnable
  [0..1] helper
  [0..1] variable
  [0..1] runnableField
    [1..1] typedRunnable
    [1..1] name (String)

(a) Expert

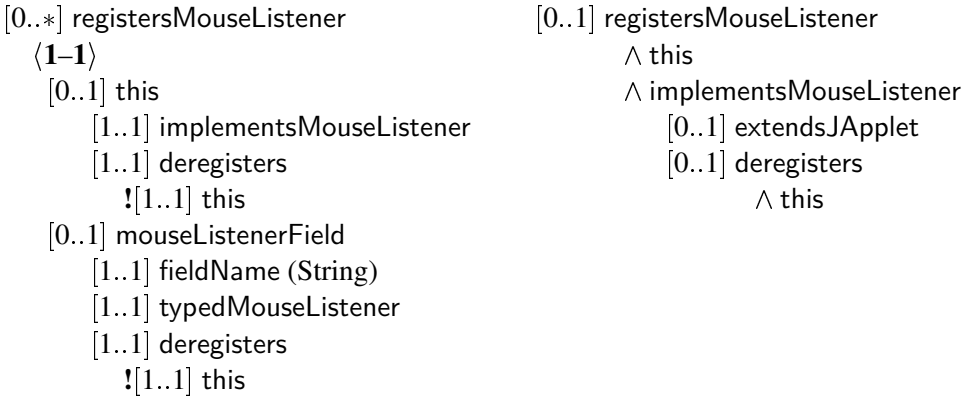
[0..1] initializesThreadWithRunnable
  ∧ this
  ∧ implementsRunnable

(b) Mined

MouseListener and extendsJApplet. Similar to the previous set of features, the mouse listener features are also missing an XOR-group in the mined feature model. The sample applets did not contain an instance of mouseListenerField. This caused the feature to be absent in the mined model, and thus, a feature group was not constructed and the this feature was collapsed into an AND-group along with one of its children, implements-MouseListener.

The deregisters feature is an optional feature in the mined model, whereas it is a mandatory feature in the expert-specified FSML model.

An interesting result of the mining process is that the extendsJApplet feature is nested under registersMouseListener. To explain this result, the sample set contained only a single class extending `JApplet` and the instance implemented a mouse listener. As a result, the exhibited relationship may be be inaccurate as a result of low support.



(a) Expert (b) Mined

4.2.3 Soft Constraints

In this section, the mined soft constraints of the applet sample set are presented and analyzed. Due to the large number of soft constraints, constraints with less than 70% confidence were filtered out. In addition, only a single constraint involving a AND-group is shown, since the other constraints can be inferred. The soft constraints are sorted by descending confidence.

Parameters and Shows Status

Soft Constraint	*supp*	*conf*
providesParameterInfo ⇒ overridesStart	21	0.84
showsStatus ⇒ overridesStart	8	0.80
showsStatus ⇒ overridesPaint	8	0.80
showsStatus ⇒ message	8	0.80
message ⇒ overridesPaint	6	0.75
providesParameterInfo ⇒ overridesPaint	13	0.72
parameter ⇒ overridesStart	23	0.70

In the feature hierarchy, parameter is nested under the overridesInit feature. The soft constraints show that parameters and the overridesStart and overridesPaint features are also highly correlated. As a result, it may be due to a biased sample set that has drawn the

conclusion that parameter is a child of overridesInit in the hierarchy. As a result, an expert may decide that parameter should simply be a child of Applet instead.

The soft constraints for showsStatus infer that there is a high confidence of being nested under overridesStart or overridesPaint. In addition, showsStatus often has a message associated with it. Looking at the mined feature hierarchy, showsStatus is nested under parameter, however, these soft constraints imply that the relationship may also be a result of a biased sample set.

Mouse Listener

Soft Constraint	*supp*	*conf*
registersMouseListener ⇒ implementsMouseListener	23	0.88
implementsMouseListener ⇒ overridesPaint	20	0.87
deregisters ⇒ parameter	8	0.80
deregisters ⇒ overridesStart	8	0.80
registersMouseListener ⇒ overridesPaint	20	0.77
deregisters ⇒ providesParameterInfo	7	0.70

In this set of soft constraints, there are a total of 26 configurations containing instances of registersMouseListener (i.e. $23/0.88 = 26$). Three sample applets registered a mouse listener, but did not implement a mouse listener. These applets, instead registered a separate inner or anonymous class that implemented the `MouseListener` interface.

In the mined feature hierarchy, the deregisters feature is a child of overridesPaint and overridesInit. In the set of soft constraints, deregister is also highly correlated with overridesStart. From this information, we can deduce that deregisters may simply be a child of Applet instead of one of the three lifecycle methods.

Threads

Soft Constraint	*supp*	*conf*
nullifiesThread ⇒ initThreadWithRunnable	16	0.94
initializesThread ⇒ initThreadWithRunnable	22	0.92
nullifiesThread ⇒ overridesStart	15	0.88
thread ⇒ initThreadWithRunnable	22	0.88
initThreadWithRunnable ⇒ overridesStart	19	0.86
thread ⇒ overridesStart	21	0.84
nullifiesThread ⇒ overridesPaint	13	0.76
initThreadWithRunnable ⇒ nullifiesThread	16	0.73
initThreadWithRunnable ⇒ parameter	16	0.73
initializesThread ⇒ nullifiesThread	17	0.71
nullifiesThread ⇒ parameter	12	0.71

The first soft constraint dealing with nullifiesThread indicates that in 16 out of the 17 configurations containing the feature, the thread was initialized using a `Runnable`. Consequently, there is only a single sample applet containing an instance where the thread was nullified, and it was initialized with a subclass. This indicates that reviewing the single applet, may be necessary to ensure that the implementation conforms to the suggested framework rules.

The second constraint initializesThread ⇒ initializesThreadWithRunnable further highlights issues with the distribution of `Runnable` and subclasses in the sample set. 92% of applets that initializes a thread, the thread was initialized using a `Runnable`. This indicates that only two applets initialized the thread using a subclass.

The overridesStart feature also commonly appears in the consequent of this set of soft constraints. This is not surprising, since a subtree of threads, the initializesThreadWithSubclass AND-group, is a child of overridesStart in the mined feature hierarchy. As a result, it can be deduced that either overridesInit or overridesStart are both likely candidates for initializing threads.

4.2.4 Study Conclusions

Feature model mining can be applied as a means of understanding framework usages from a sample set of applications. The mined metamodel shown in the previous section has provided insight into the use of the applet framework. In addition, these observations can be used to update the expert-specified metamodel to better represent the framework.

In conclusion, the mined applet metamodel captures many of the relationships that are modeled in the expert-specified metamodel. However, there are several notable defects with the mined metamodel. First, a set of mandatory features are collapsed into a single AND-group, which would need to be separated out by the user post-mining. Next, the mining algorithm is not tolerant of any errors in the dataset. Evidence of this deficiency is with the mined nullifiesThread and the overridesRequiredMethods OR-group. Next, the mined metamodel is in the generalized feature model notation, and user input is required in order to construct a true feature hierarchy. Finally, the mining algorithm is very susceptible to relationships with low support or none at all. Feature which did not appear in the sample set were not represented at all in the mined feature model. These issues and suggested solutions are described further in Chapter 5.

4.3 Mining on the Framework Boundary

The third use case we identified is the application of feature model mining to jump-start FSML development. In this scenario, an existing FSML mapping definition is not available to construct framework-specific models. As a result, the mapping between code patterns and framework-provided features must also be mined prior to constructing a sample set.

A suggested approach for solving this problem is mining on the framework boundary. The framework boundary consists of all calls that occur between an application and the framework. The boundary includes both callback methods implemented in application code, as well as framework methods called via the framework API.

Pre-defined code queries can approximate the framework boundary by identifying the method calls to the framework and callback methods implemented by an application. The results of these code queries on a sample application are used to construct a sample configuration. Thus, the sample set of configurations is constructed by executing the code queries on a set of sample applications.

While pre-defined queries are sufficient for constructing a useful sample set, user-specified queries provide additional flexibility. A user-specified query allows users to focus the mining procedure on a particular framework subsystem, or add additional facts to the sample set. The use of user-defined queries is further discussed in Section 5.2.

Once the sample set is constructed, the mining algorithm can be used to retrieve a feature model using the procedure described in Chapter 3.

CHAPTER 5

Analysis and Future Work

In the previous chapter, a feature model representing the Java applet framework was reverse-engineered from a set of sample applications. The study identified several issues with the mining algorithm. In this chapter, recommendations and extensions to the feature model mining algorithm are discussed and suggested solutions are presented. The extensive list of improvements to the mining algorithm opens up a large area of research.

5.1 Cardinality-Based Feature Models

The mining algorithm presented in this thesis is capable of constructing only basic feature models. Mining for cardinality-based feature models was not addressed by the algorithm.

The current algorithm mines for optional features $[0..1]$, and AND-groups which represent groups of features with cardinality $[1..1]$. Other cardinalities, such as $[0..*]$, have not been addressed. For feature groups, the current mining algorithm retrieves OR-groups $\langle 1..n \rangle$ and XOR-groups $\langle 1..1 \rangle$. Methods for mining more general $\langle m..n \rangle$ feature groups should be investigated as future work.

Addressing this issue will require changes to both the feature model synthesis algorithm as well as the mining procedure. The synthesis algorithm constructs a feature model from a propositional formula and will need to be extended in order to handle feature cloning. An issue to consider is that a propositional formula can not represent multiplicity, or a multiset of variables. This work will need to be done prior to the construction of a cardinality-based feature model from logic.

5.2 User-specified Queries

The sample set is currently constructed from queries specified by an expert in either an FSML mapping definition (Section 4.2), or through a set of pre-defined framework

boundary queries (Section 4.3). However, the user currently has no control over how the sample set of configurations is constructed.

User-specified queries provide users with the capability of specifying their own set of queries to construct a sample set. For example, user-specified queries can be applied by a user to focus the mining algorithm on a particular subsystem in a large framework, such as a single type. Also, obscure code patterns that the user is familiar with may not be picked up by an automatic miner.

Presenting an interface for constructing queries is a challenging task. The interface must be powerful enough to specify complex queries, yet require less effort than specifying an entire FSML mapping definition. Constructing an interface for composing the already existing code queries used in the automatic extraction of framework-specified models [AC06] seems like a good starting point.

Constraint-based mining can be used to implement such queries. Constraint-based mining allows the user to specify expectations as constraints to confine the search space [HK00].

5.3 Correcting Errors in the Sample Set

The mining algorithm presented in this thesis constructs a feature model representative of a sample set of configurations. However, if the configurations contained errors, then the errors would be represented in the mined feature model. In addition, the sample set is typically an incomplete representation. As a result, the algorithm may derive incorrect relations among its features especially among relationships with low support. This section describes the following: (i) the effect that an error has on the different components of a feature model, (ii) possible methods for dealing with such errors, (iii) methods for presenting suggestions for corrections, and (iv) filtering rules from the set of soft constraints.

5.3.1 Effect of Errors on the Feature Hierarchy

In the presented algorithm, only association rules with 100% confidence are selected as hard constraints. As a result, the mined feature model is very sensitive to errors in the sample set. In this section, a preliminary analysis of the effect that erroneous sample configurations or an incomplete sample set will be discussed.

Feature hierarchy. A binary implication $f_i \rightarrow f_j$ will be absent from the set of hard constraints in the case where the consequent of the implication, f_j, is absent when the antecedent, f_i, is present in a sample configuration. The absence of an expected binary implication will cause either, (i) the feature f_i to be absent in the hierarchy if there no other binary implications, or (ii) the feature f_i to appear higher in the hierarchy than expected.

The presence of an erroneous binary implication will cause an entire sub-tree of features to be shifted in the mined feature model. For example, the mined applet metamodel in Section 4.2 (Figure 4.3) has a much deeper nesting hierarchy than the expert-specified model (Figure 4.1). The additional nesting was introduced by the presence of additional binary implications. These implications were gathered from the patterns exhibited from the sample set, and may, in fact, have produced a feature model that is more constrained than necessary.

AND-groups. An AND-group is constructed when bi-implication exists among a set of features. Thus, a feature f_i will be present in an AND-group $f_1, \ldots, f_k$ if all f_i features have bi-implications among each other.

In the case where a bi-implication is not present for a feature due to an incomplete sample set, the feature will remain as an optional feature and appear as sibling of the AND-group in the feature hierarchy. On the other hand, an additional, erroneous bi-implication will cause an optional feature to collapse into a AND-group.

Group implications. A group implication, $f \rightarrow f_1 \vee \cdots \vee f_k$ constructs an OR-group f with subfeatures $f_1, \ldots, f_k$ given that its subfeatures are descendants of the group feature f. An invalid configuration where f is present, and any one of its subfeatures $f_1, \ldots, f_k$ are absent will cause a group implication to be absent, and thus, the OR-group.

The subfeatures, $f_1, \ldots, f_k$, will appear simply as a set of optional features under the parent f. Erroneous group implications and thus, additional OR-groups may appear in the mined model due to an incomplete sample set.

Mutual exclusion clauses. An XOR-group f with subfeatures $f_1, \ldots, f_k$ is constructed with a set of mutual exclusion clauses $f_i \rightarrow \overline{f_j}$ for all combination of subfeatures, in addition to the constructs of an OR-group. If any one of the mutual exclusion clauses is absent due to an error in the sample set, then the feature group would remain as an OR-group.

An incomplete sample set can generate erroneous mutual exclusion clauses. A single erroneous mutual exclusion clause will have no effect on the mined feature model. Only in the case where all k^2 subfeatures of a feature group $f \rightarrow f_1 \vee \cdots \vee f_k$ are mined, will an expected OR-group be further constrained into a XOR-group.

5.3.2 Dealing with Errors

The feature hierarchy and additional hard constraints in the mined feature model are constructed from association rules with 100% confidence. However, in the event that an error exists in the sample set, valid hard constraints will have less than 100% confidence. These expected hard constraints will instead be contained in the set of soft constraints. Two methods for retrieving these valid constraints are described. First, the confidence threshold

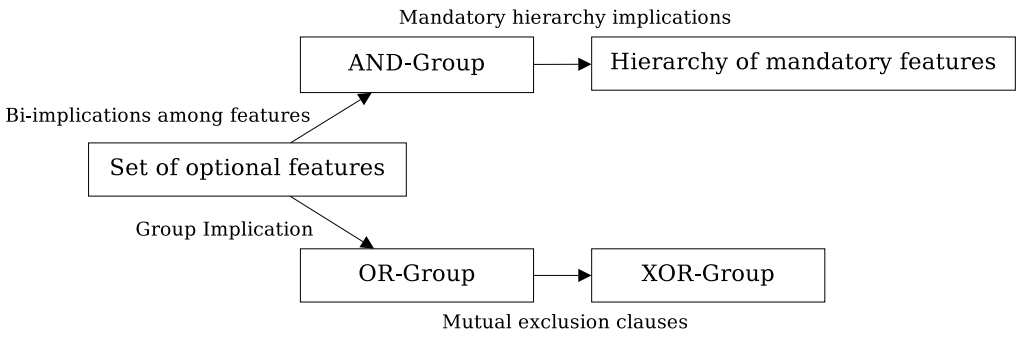


Figure 5.1: Progression of a set of constraints

for hard constraints can be lowered. Second, an interactive approach for promoting and demoting constraints is described.

Lowering the confidence threshold. A common approach for dealing with sample set errors is to lower the minimum confidence threshold when mining for association rules [BSM06, Mic00]. The feature model mining algorithm uses a strict confidence threshold of 100% when constructing the feature hierarchy. Using association rules with less than 100% confidence may retrieve valid constraints from the set of soft constraints.

A side effect of including constraints with less than 100% confidence in the feature hierarchy is that both soft constraints and hard constraints are present in the same model. As a result, it is important to separate the hard constraints from the soft constraints when presenting such a feature hierarchy. A suggestion would be to annotate the feature types and subfeature relationships with the mined confidence and support measures similar to the notation used in a probabilistic feature model [CSW08]. However, since feature types such as OR-groups and XOR-groups are constructed using a number of different formulas, it is unclear how best to present the different confidence and support measures for each of the formulas.

Interactive approach. The previous approach of lowering the confidence threshold is likely to add false positives to the mined feature hierarchy. An interactive approach is complementary. The user may still wish to modify the mined feature model using information from the set of soft constraints.

A proposed method for modifying the mined feature model is based on a progression of feature types where a user can *promote* or *demote* a feature or feature group. In Figure 5.1, an example progression between the different feature types in the feature hierarchy is shown. At the lowest level is a set of sibling optional subfeatures, $f_1, \ldots, f_k$, with parent feature, f. The optional subfeatures were constructed using a set of binary implications, $f_i \rightarrow f$ where $1 \leq i \leq k$.

There are two possible paths of progression at this point. Given that the set of features are connected through a set of bi-implications where $f_i \rightarrow f_j$ where $1 \leq i, j \leq k$ exist among all of the optional features, an AND-group is constructed.

Alternatively, given a group implication $f \rightarrow \bigvee_{1 \leq i \leq k} f_i$, an OR-group will be created. The OR-group can be further constrained as a XOR-group if mutual exclusion clauses, $f_i \veebar f_j$ where $1 \leq i, j \leq k$, are present as hard constraints.

A constraint is promoted by adding the additional constraints to the feature hierarchy formula in order to construct the next type of feature. A constraint is demoted by removing the corresponding constraint. For example, when promoting a set of optional features to an OR-group, an additional group constraint is added to the feature hierarchy formula. When demoting an XOR-group to an OR-group, the mutual exclusion clauses are removed.

5.3.3 Presenting Constraints

A method for presenting suggested changes to the feature hierarchy will need to be researched as part of future work. As a general rule, the soft constraints with the highest confidence should appear first in the list of possible changes. Rules should be presented in terms of its effect on the feature hierarchy rather than arbitrary propositional formulas. For example, adding a group implication and mutual exclusion clauses changes a set of optional features to an XOR-group. This change should be presented as an atomic operation instead of the addition of multiple constraints.

However, adding or moving a feature in the feature hierarchy may have side effects which affect its subfeatures and these side effects should be presented to the user as well. Investigating possible methods of presenting these constraints should be done in the future.

5.3.4 Filtering Soft Constraints

Soft constraints are presented to the user as a set of constraints sorted by descending confidence as shown in Figure 3.5 on page 27. The set of soft constraints may contain a large number of redundant or misleading rules that could be filtered prior to presenting the constraints to the user.

Subsumption is logical property that can be used to determine redundant rules. A Boolean expression A subsumes B if A logically implies B. For example, the rule (i) $Applet \rightarrow Stop \wedge Start$ subsumes the rules (ii) $Applet \rightarrow Stop$, and (iii) $Applet \rightarrow Start$. In the scenario where all of these rules are boolean implications (i.e. are found with 100% confidence), it is not necessary to present rules (ii) and (iii) to the user since rule (i) present the same information in a more concise manner. However, in the case where the rules have different confidence, there are more intricate rules that exist for filtering unnecessary rules. Implementing this form of filtering is a useful extension to the mining algorithm.

Several methods for filtering association rules are described by Michail [Mic00] and used in FrUiT [BSM06]. Further research in applying association rule filtering in the context of feature model mining is a part of future work.

5.4 Using Prior Knowledge

Often times, a user may have knowledge regarding the structure of the mined feature model. This knowledge can be provided before or after the mining process in order to refine the mined feature model. Providing additional knowledge to the mining algorithm will yield a more precise and useful feature model to the user.

Three uses of prior knowledge are identified: (i) first, prior knowledge can be used to separate additional hard constraints from the set of constraints used to construct the feature hierarchy, (ii) user input can separate an AND-group into a hierarchy of mandatory features and (iii) the user may have an idea of the resulting feature model, and may wish to assert such a model a priori. These three uses will be discussed in the following subsections.

5.4.1 Additional Hard Constraints

The feature hierarchy formula is a conjunction of formulas used in constructing the hierarchy of the mined feature model. As described in Section 3.4, only group implications which cut across the feature hierarchy are separated as additional constraints. For example, in the mined clock feature model (shown in Figure 3.7 on page 25), the group implication $Clock \rightarrow Alarm \vee 24hr \vee (Analog \wedge Hands)$ was presented as an additional hard constraint because 24hr was not an immediate child of Clock. All other types of implications are added to the feature hierarchy formula.

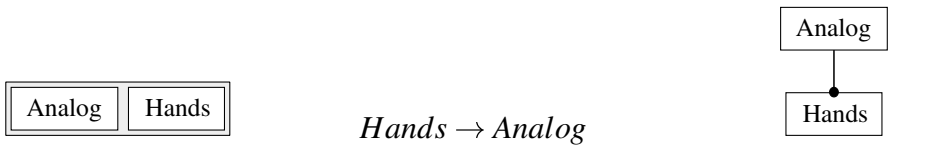
Additional hard constraints can also be used to represent hard constraints that have been removed from the feature hierarchy. When a user modifies the feature model post mining using an interactive approach as described in Section 5.3.2, constraints can also be moved from the hierarchy to the set of additional hard constraints. Doing so will maintain the logical constraints that the feature model imposes on its legal configurations.

A common task for users will be to remove the multiple parent relations from a generalized feature model which is represented as a DAG to form a tree. In Figure 4.3 on page 37, the deregisters AND-group has two parent features, overridesPaint and the registersMouseListener AND-group. If the registersMouseListener feature was selected to be the parent feature in the hierarchy, the child-parent relation $deregisters \rightarrow overridesPaint$ can be preserved as an additional constraint.

5.4.2 Separating AND-groups

The feature model synthesis algorithm constructs AND-groups from the set of bi-implicated features in the feature hierarchy formula. However, the current algorithm does not separate the AND-group into a hierarchy of mandatory features. An extension to the algorithm is to separate the features in an AND-group using additional knowledge.

A proposed means of separating AND-groups is to specify a set of implications among features in an AND-group. The implications, called *mandatory hierarchy relations*, deter-



(a) An AND-group (b) Mandatory hierarchy relations (c) Separated mandatory features

Figure 5.2: Separating AND-groups

mine the nesting relationship that the grouped features will have when it is separated into a set of mandatory features.

In Figure 5.2a, an AND-group consisting of two features, Analog and Hands is shown. The AND-group can be separated into two mandatory features with a separate mandatory hierarchy relation $Hands \rightarrow Analog$. As a result, Hands is a subfeature of Analog. The resulting mandatory feature hierarchy is shown in Figure 5.2c.

5.4.3 Asserted Structures

One of the goals of feature model mining is to assist in the evolution of domain-specific languages. Consequently, often times, a feature model representing the domain may already exists. Such is the case with FSMLs. We refer to this existing model as the *asserted model* hereafter. Feature model mining could be used to incorporate new concepts or relations into this asserted model.

In Chapter 4, feature model mining was applied to retrieve a model to refine or discover new patterns in an object-oriented framework. An existing feature model existed in the form of an FSML metamodel. Analyzing and merging the differences between the mined metamodel and the expert-specified model is currently a manual task.

The asserted model does not need to be a complete representation of the domain. Partial relationships still add useful knowledge to the mining algorithm. For example, in Java, we know that a class is the parent of a method. Thus, in the mined feature model, a feature such as *overridesStart* should be a child of extendsApplet. This knowledge could be specified as part of an propositional formula, called an *asserted formula*, prior to the start of the mining algorithm.

The asserted formula should contain the facts and relationships which the user guarantees, or asserts to be $true$. Thus, the mining algorithm should incorporate such knowledge into the mined feature model if it does not contradict the model. However, any mined constraints that contradict the asserted formula should be brought to the user's attention. For example, a warning should be issued in the case where an asserted formula contained $showsStatus \rightarrow message$ and mined constraint had less than 50% confidence. The asserted formula can then serve as a mechanism of validating and testing certain assumptions of the domain on the sample set.

The concept of an asserted formula should be further investigated as part of future work.

5.4.4 Other Sources of Knowledge

The current research has been focused on mining feature models from the source code of a set of sample framework applications. Source code is only one knowledge artifact in a typical object-oriented framework. Other sources of knowledge include framework documentation, code comments and existing models. The knowledge contained in these artifacts can be added to the sample set if an appropriate mapping definition or interpreter was written. Incorporating these sources of knowledge may increase the precision of the mining algorithm.

5.5 Rule Mining Optimizations

Feature model mining uses both conjunctive and disjunctive association rule mining to retrieve the different forms of expressions used to construct a feature model. The basic conjunctive association rule mining is implemented as described by Agrawal et al. [AIS93]. In this section, several improvements to basic mining procedure are described.

5.5.1 Alternate Interestingness Measures

The interestingness measures used in this thesis, support and confidence, as discussed in Section 3.3, are not the only interestingness measures available for an association rules. Other interestingness measures include *lift* (originally called *interest*) [BMS97] and *conviction* [BMUT97].

The lift measure [BMS97] measures the correlation between the antecedent and the consequent of an association rule using the χ^2 test. The chi-squared test specifies whether all k items are k-way independent. It is a bi-directional measure, such that the rules $A \Rightarrow B$ and $B \Rightarrow A$ will have the same lift. Lift is defined in Equation 5.1. Lift has an advantage over confidence in that it incorporates $P(B)$ into the interestingness measure.

$$\text{lift}(A \Rightarrow B) = \frac{|\mathcal{S}| \times supp(A \wedge B)}{supp(A) \times supp(B)} = \frac{P(B|A)}{P(B)} = \frac{P(A,B)}{P(A)P(B)} \tag{5.1}$$

Conviction is another measure from Brin et al. [BMUT97]. Unlike lift, conviction is directional and measures actual implication rather than the correlation. Conviction is defined for an association rule $A \Rightarrow B$ with a sample set $\mathcal{S}$ as:

$$\text{conv}(A \Rightarrow B) = \frac{supp(A) \times supp(\overline{B})}{|\mathcal{S}| \times supp(A \wedge \overline{B})} = \frac{P(A)P(\overline{B})}{P(A,\overline{B})} \tag{5.2}$$

Association rules in which A and B are completely unrelated have a conviction of 1 and rules that always hold have a conviction value of ∞. However, in practice, to avoid dealing with infinities, the conviction measure can be inverted such that implications have a value of zero.

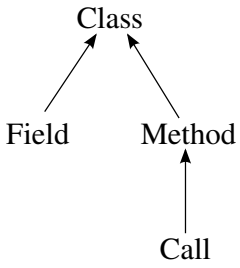


Figure 5.3: Simple taxonomy

A survey of many other interestingness measures for association rules was done by Lenca et al. [LVML07]. The various interestingness measures could be used as alternative methods of presenting soft constraints to the user.

5.5.2 Generalized Association Rules

Generalized association rules are a form of conjunctive association rules first introduced by Srikant and Agrawal [SA95]. An associated taxonomy is provided to the mining algorithm.

The primary benefits of generalized association rules is in the form of speed improvements for the mining algorithm and in rule filtering. A smaller set of rules will be presented to the user because generalized association rule mining can retrieve the most specific form of rule that satisfies the minimum interestingness measures. Michail [Mic00] uses the class inheritance structure as its taxonomy when mining for generalized association rules.

Generalized association rule mining does not discover additional rules when compared with traditional association rule mining. The associated taxonomy can be flattened such that the ancestors of each feature are added to the each sample configuration. For example, given the taxonomy shown in 5.3, the configuration $\{\text{Field}, \text{Call}\}$ is equivalent to the transaction $\{$Field, Call, Method, Class$\}$. Using this technique, a traditional association rule miner can find the same association rules as a generalized association rule miner.

CHAPTER 6

Related Work

6.1 API Usage Mining

In the context of understanding framework usages, CodeWeb and FrUiT are two notable works which use association rule mining on a set of sample applications.

CodeWeb [Mic00] mines for API usage patterns from a sample set of applications using association rule mining. The patterns can be viewed through a browser interface. However, unlike the algorithm presented in this thesis, CodeWeb does not synthesize a model.

FrUiT [BSM06] is an IDE tool which provides assistance in the form of framework completion suggestions. FrUiT combines data mining with context-dependent presentation. FrUiT's context is limited to the current Java class and rules are sorted by confidence. In comparison, feature model mining is a generic mining approach which constructs a model of variability out of any sample set.

FUDA [HBC07] is an approach developed by fellow members of the Generative Software Development Lab. FUDA constructs implementation recipes for framework concepts using code analysis and data mining techniques. However, the focus of FUDA and feature model mining are different. FUDA collects traces from multiple executions of an application using dynamic analysis. The recipe constructed by FUDA describes the detailed implementation steps for a single feature in a framework. In contrast, feature model mining synthesizes a model describing the relation between the features in a framework using static analysis.

6.2 Ontology learning systems

As Ontologies and feature models are related forms of domain modeling [CKK06], ontology learning systems [SB03] are related to feature model mining. Ontology learning

systems are systems which extract ontological elements from input and construct a model representing the concepts described. Both ontology learning systems and feature model mining synthesize a model using a sample set of facts.

Ontology learning systems typically deal with natural language processing. As a result, the relationship and concepts in an learned ontology is retrieved from the syntactic and semantic properties of the input language. A key difference between ontology learning systems and feature model mining lies in what the synthesized model describes. A learned ontology models concepts and relationships as described within the input data. A mined feature model, on the other hand, models the variability of the input data itself. An analogy between the systems is that ontology learning systems are concerned with learning the semantics of a sentence, while feature model mining is concerned with understanding the grammar.

6.3 Bayesian network learning

Bayesian learning tools [MJKL05, JN07] attempts to construct a Bayesian network with a joint probability distribution that adequately describes the sample set of data. Bayesian networks and probabilistic feature models are very similar in nature as shown in the paper by Czarnecki et al. [CSW08]. However,concepts in a Bayesian network are represented as nodes in a directed acyclic graph. In a feature model, the concepts are part of a hierarchy (tree). As a result, the learned Bayesian network may be more difficult to interpret than a feature model.

However, many similar problems exist between Bayesian learning and feature model mining such as (i) dealing with a biased or flawed sample set, (ii) determining how closely the learned / mined model should be to the sample set and (iii) choosing an appropriate structure for the model. The similarities between feature model mining and Bayesian learning should be further explored.

CHAPTER 7

Conclusions

The feature model mining algorithm was presented in this thesis. Feature model mining provides an analytical tool for reconstructing a model of variability from a sample set of configurations. Applied to systems families, feature model mining can retrieve a model for a family from a sample set of software systems and applications. As a practical example, the mining algorithm is applied to aid in the development of a framework-specific modeling language (FSML), used to describe the domain of an object-oriented framework.

The core of the mining algorithm is in finding propositional formulas representing the different components of a probablistic feature model from the input sample set. In this thesis, the mining of (i) binary implications, (ii) group implications, and (iii) mutual exclusion clauses are discussed. The mined implications are used to construct the feature hierarchy. Retrieving the set of additional hard constraints and soft constraints are also discussed.

Applying the algorithm to a real-world scenario, the mining algorithm was used to mine for a metamodel representing the Java applet framework. From the experiment, the algorithm is shown to be capable of describing framework usages from a sample set of applications. The mined metamodel was useful in providing insight on the use of the applet framework, and the set of soft constraints show patterns in the framework applications which may be useful for posterior editing of the feature model or for framework comprehension.

The work presented in this thesis is one step towards a method for reverse-engineering a feature model from a sample set. Several extensions to the algorithm include (i) cardinality-based feature models, (ii) methods for dealing with sample set error and bias, (iii) using prior knowledge, (iv) using other sources of knowledge and (v) rule mining optimizations.

Feature model mining is a tool in the generative software development paradigm. Generative software development seeks to automate development of a software system family by generating a software system through the use of a specification written using domain-specific languages. These languages are currently written using a top-down methodology

where an expert analyzes domain artifacts and constructs an appropriate model of the domain. In addition, the software systems may evolve over time and the language must also evolve to remain consistent. Feature model mining addresses this problem by providing a bottom-up approach for constructing a model. The mining algorithm reverse-engineers a specification through analyzing the variability present in a sample set of systems.

Appendix A: Applet Sample Set

Gathered from Sun Tutorials

```
Animator
ArcTest
BarChart
Blink
Clock
DitherTest
DrawTest
Fractal
GraphicsTest
GraphLayout
ImageMap
JumpingBox
MoleculeViewer
NervousText
SimpleGraph
SortDemo
SpreadSheet
TicTacToe
WireFrame
```

Gathered from the Internet

```
anbutton
antacross
antmarch
aqua
blinkhello
brokeredchat
bsom
buttontest
camk
client
consultomatic
cte
demographics
dotproduct
envelope
fireworks
formeln
gammabutton
geometry
hellotcl
hyperbolic
iagttager
inspect
jscriptex
kbdfocus
linprog
mousedemo
myapplet2
nickcam
notprolog
scatter
scope
simplepong
simplesun
smtp
starfire
superapplet
swatch
tetris
urcrccalendar
urlexample
vechat
webstart
ympyra
```

Acknowledgements. Thanks to Michał Antkiewicz for constructing this sample set used in our case study [App08].

Bibliography

[ABC07] M. Antkiewicz, T. T. Bartolomei, and K. Czarnecki, "Automatic extraction of framework-specific models from framework-based application code," in *ASE*, R. E. K. Stirewalt, A. Egyed, and B. Fischer, Eds. ACM, 2007, pp. 214–223. 10, 31, 35

[AC06] M. Antkiewicz and K. Czarnecki, "Framework-specific modeling languages with round-trip engineering," in *MoDELS '06*. Springer, 2006, pp. 692–706. 2, 10, 31, 46

[AC07] M. Antkiewicz and K. Czarnecki, "Framework-specific modeling languages; examples and algorithms," ECE, Univeristy of Waterloo, Tech. Rep. 2007-18, 2007. [Online]. Available: http://gp.uwaterloo.ca/tr/2007-TR-antkiewicz-fsmls.pdf 2, 30, 32, 34

[AIS93] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD '93*. New York, NY, USA: ACM, 1993, pp. 207–216. 10, 15, 16, 19, 25, 28, 29, 30, 52

[Ant08] M. Antkiewicz, "Framework-specific modeling languages," Ph.D. dissertation, University of Waterloo, Waterloo, Ontario, Canada, 2008. 31, 35

[App08] "Listing of sample applets," Generative Software Development Lab, 2008. [Online]. Available: http://gsd.uwaterloo.ca/projects/fsmls/applet-fsml/applet-examples/ 58

[AS94] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 487–499. 16

[ATT08] "GraphViz," AT&T Research, 2008. [Online]. Available: http://www.graphviz.org 30

[Bat05] D. S. Batory, "Feature models, grammars, and propositional formulas," University of Texas at Austin, Texas, Tech. Rep. TR-05-14, Mar. 2005. 6

[BBC06] D. S. Batory, D. Benavides, and A. R. Cortés, "Automated analysis of feature models: challenges ahead," *Commun. ACM*, vol. 49, no. 12, pp. 45–47, 2006. 6

[BBS04] R. I. Bull, C. Best, and M.-A. Storey, "Advanced widgets for eclipse," in *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*. New York, NY, USA: ACM, 2004, pp. 6–11. 30

[BMS97] S. Brin, R. Motwani, and C. Silverstein, "Beyond market baskets: Generalizing association rules to correlations," in *SIGMOD Conference*, J. Peckham, Ed. ACM Press, 1997, pp. 265–276. 52

[BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," in *SIGMOD Conference*, J. Peckham, Ed. ACM Press, 1997, pp. 255–264. 52

[Bry86] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, Aug. 1986. [Online]. Available: http://www.cs.cmu.edu/~bryant/pubdir/ieeetc86.ps 28

[BSM06] M. Bruch, T. Schäfer, and M. Mezini, "Fruit: Ide support for framework understanding," in *eclipse '06*. New York, NY, USA: ACM, 2006, pp. 55–59. 48, 49, 54

[CAK+05] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek, "fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates," in *OOPSLA Companion*, R. Johnson and R. P. Gabriel, Eds. ACM, 2005, pp. 200–201. 5, 38

[CE00] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston, MA: Addison-Wesley, 2000. 1, 4, 5

[CHE05] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged configuration through specialization and multilevel configuration of feature models," *Software Process: Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005. 5, 10

[CK05] K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: a progress report," in *International Workshop on Software Factories*, 2005. 35

[CKK06] K. Czarnecki, C. H. P. Kim, and K. Kalleberg., "Feature models are views on ontologies," in *Proceedings of 10th International Software Product Line Conference (SPLC 2006)*. IEEE, 2006, pp. 41–51. 54

[CSW08] K. Czarnecki, S. She, and A. Wąsowski, "Sample spaces and feature models: There and back again," in *SPLC '08*. IEEE, Sep. 2008, to appear. 8, 29, 48, 55

[CW07] K. Czarnecki and A. Wąsowski, "Feature models and logics: There and back again," in *SPLC '07*. IEEE, Sep. 2007. 11, 12, 27, 28, 29, 30

[Cza04] K. Czarnecki, "Overview of Generative Software Development," in *Proceedings of Unconventional Programming Paradigms (UPP) 2004, 15-17 September, Mont Saint-Michel, France, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 3566. Springer-Verlag, 2004, pp. 313–328. [Online]. Available: http://www.swen.uwaterloo.ca/~kczarnec/gsdoverview.pdf 1

[Ecl08a] "Eclipse IDE," Eclipse Foundation, 2008. [Online]. Available: http://www.eclipse.org/ 30

[Ecl08b] "Eclipse java development tools (JDT) subproject," Eclipse Foundation, 2008. [Online]. Available: http://www.eclipse.org/jdt/ 30

[HBC07] A. Heydarnoori, T. T. Bartolomei, and K. Czarnecki, "Comprehending Object-Oriented Software Frameworks API Through Dynamic Analysis," School of Computer Science, University of Waterloo, Technical Report CS-2007-18, October 2007. 54

[HK00] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000. 9, 16, 18, 19, 46

[JN07] F. V. Jensen and T. D. Nielsen, *Bayesian Networks and Decision Graphs*. Springer, 2007. 55

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Technical Report CMU/SEI-90-TR-21, 1990. 4, 5

[Lau06] S. Q. Lau, "A domain analysis of e-commerce systems using feature-based model templates," Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2006. 1

[LKL02] K. Lee, K. C. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering," in *Software Reuse: Methods, Techniques, and Tools: Proc. of the Seventh Reuse Conference (ICSR7), Austin, USA, Apr. 15-19, 2002*, ser. Lecture Notes in Computer Science, C. Gacek, Ed., vol. 2319. Heidelberg, Germany: Springer-Verlag, 2002, pp. 62–77. 1

[LVML07] P. Lenca, B. Vaillant, P. Meyer, and S. Lallich, "Association rule interestingness measures: Experimental and theoretical studies," in *Quality Measures in Data Mining*, ser. Studies in Computational Intelligence, F. Guillet and H. J. Hamilton, Eds. Springer, 2007, vol. 43, pp. 51–76. 53

[Mic00] A. Michail, "Data mining library reuse patterns using generalized association rules," in *ICSE '02*. ACM, 2000, pp. 167–176. 48, 49, 53, 54

[MJKL05] A. L. Madsen, F. Jensen, U. Kjærulff, and M. Lang, "The Hugin tool for probabilistic graphical models," *International Journal on Artificial Intelligence Tools*, 2005. 55

[NCJK01] A. A. Nanavati, K. P. Chitrapura, S. Joshi, and R. Krishnapuram, "Mining generalised disjunctive association rules," in *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*. New York, NY, USA: ACM, 2001, pp. 482–489. 21, 23

[Obj04] *Model-Driven Architecture*, Object Management Group, 2004. [Online]. Available: http://www.omg.org/mda 31

[ope08] "xText," openArchitectureWare.org, 2008. [Online]. Available: http://www.xtext.org 30

[Pec97] J. Peckham, Ed., *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*. ACM Press, 1997.

[SA95] R. Srikant and R. Agrawal, "Mining generalized association rules," in *VLDB*, 1995, pp. 407–419. 53

[SB03] M. Shamsfard and A. A. Barforoush, "The state of the art in ontology learning: a framework for comparison," *Knowl. Eng. Rev.*, vol. 18, no. 4, pp. 293–316, 2003. 54

[Sch06] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006. 31

[Ste99] G. L. Steele Jr., "Growing a language," *Higher-Order and Symbolic Computation*, vol. 12, no. 3, pp. 221–236, 1999. 1

[Sun08] "Applets," Sun Microsystems, Inc., 2008. [Online]. Available: http://java.sun.com/applets/ 32

[UAUA04] T. Uno, T. Asai, Y. Uchida, and H. Arimura, "An efficient algorithm for enumerating closed patterns in transaction databases," in *Discovery Science*, ser. Lecture Notes in Computer Science, E. Suzuki and S. Arikawa, Eds., vol. 3245. Springer, 2004, pp. 16–31. 18

[UKA04] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets," in *FIMI*, 2004. 19, 30

[ZZR06] L. Zhao, M. J. Zaki, and N. Ramakrishnan, "Blosom: A framework for mining arbitrary boolean expressions over attribute sets," Technical Report 06-05, 2006, available from http://www.cs.rpi.edu/research/pdf/06-05.pdf. 21, 22, 28, 29, 30