Relating Requirements and Feature Configurations: A Systematic Approach

Tun, Thein Than; Boucher, Quentin; Classen, Andreas; Hubaux, Arnaud; Heymans, Patrick

Published in:
Proceedings of the 13th International Software Product Lines Conference (SPLC'09), San Francisco, CA, USA

Publication date:
2009

Document Version
Early version, also known as pre-print

Relating Requirements and Feature Configurations: A Systematic Approach

Thein Than Tun Quentin Boucher Andreas Classen Arnaud Hubaux Patrick Heymans
PReCISE Research Centre
Faculty of Computer Science
University of Namur, Belgium
{ttu, qbo, acs, ahu, phe}@info.fundp.ac.be

Abstract

A feature model captures various possible configurations of products within a product family. When configuring a product, several features are selected and composed. Selecting features at the program level has a general limitation of not being able to relate the resulting configuration to its requirements. As a result, it is difficult to decide whether a given configuration of features is optimal. An optimal configuration satisfies all stakeholder requirements and quantitative constraints, while ensuring that there is no extraneous feature in it. In relating requirements and feature configurations, we use the description of the problem world context in which the software is designed to operate as the intermediate description between them. The advantage of our approach is that feature selection can be done at the requirements level, and an optimal program level configuration can be generated from the requirements selected. Our approach is illustrated with a real-life problem of configuring a satellite communication software. The use of an existing tool to support our approach is also discussed.

1. Introduction

A software product family provides several features, subsets of which may be configured into products. Feature selection is concerned with the issue of determining features that should be included in a product in order to satisfy requirements expressed by various stakeholders [10, 26]. Typically, there are several sources of constraints that affect feature selection.

Some of these constraints may come from *implementation platform or language*: for instance, the use of a single-threaded operating system may prevent composition of features which require multi-threading. Some constraints may come from *requirements*: for instance, a user requirement for two-way transmission between a ground user and a spacecraft requires selection of features for "uplink" and "downlink". Some constraints may come from *the problem world context*, i.e. from the physical context in which the software is used. For instance, the fact that some users will send or receive files to a spacecraft remotely (away from a network control centre) may require inclusion of remote login features.

Although feature configuration has to happen ultimately at the program level, focusing on the code, without sufficient attention to the requirements and the problem world context, creates several difficulties [25]. In cases where resources—such as CPU time and memory capacity—are limited, inclusion of extraneous features needs to be avoided, while ensuring that the configuration satisfies all stakeholder requirements. We call this an "optimal configuration" of features. In this paper, we propose a systematic approach to generating feature configurations from selected requirements.

We use the Jackson-Zave framework to requirements engineering [35, 17] as our conceptual basis. One main principle of this framework is a systematic separation of system descriptions into *(1) requirements*, *(2) problem world context* and *(3) specifications* (features). Following this framework, we express differences in requirements, problem world context and the software, of products within a product line using three separate feature models (other alternative variability models such as OVM or decision models may be used instead). The first feature model (FM) represents different sets of stakeholder requirements that need to be satisfied in different products, the second FM represents the different contextual settings in which the software may be used, whilst the third FM represents the different possible configurations of the software. In addition, quantitative constraints, such as on the memory consumption of a configuration, are expressed.

Separation of FMs in this way allows the developers to consider feature configurations at the level of stakeholder requirements. Our approach can be used to generate the optimal configuration(s) for selected requirements, thus providing a way of relating requirements to feature configura-

tions. We demonstrate the proposed approach with the configuration of a file delivery protocol CFDP used by a satellite communication software company. An off-the-shelf tool-support is presented, and its limitations discussed. The main contribution of the paper is a systematic approach to relate requirements to feature configurations in order to obtain optimal feature configurations.

The remainder of the paper is organised as follows. Section 2 discusses the background to our work. The quantitative constraints are introduced in Section 3. The proposed approach to relating requirements to feature configuration is presented in Section 4. Application of the approach to the motivating example, and the use of an off-the-shelf tool to support our approach are illustrated in Section 5. Related work is discussed in Section 6. Section 7 provides some concluding remarks.

2 Background

This section describes the motivating example used in this paper, an overview of the Jackson-Zave framework, and the way FMs will be used.

2.1 Movivating Example

The CCSDS File delivery Protocol (CFDP), defined by the Consultative Committee for Space Data Systems (CCSDS) [9], provides the protocol to transfer files between file systems across interplanetary distances. The protocol is independent from the data storage implementation and requires only a few basic file storage capabilities. It also assumes a minimum of two filestores, typically one within the spacecraft and another on the ground. Capabilities of the protocol include deferred transmission (if the communication link is unavailable, the transfer will be performed at the next transmission opportunity), concurrent transfer and transfer suspend/resume.

Figure 1 is an example of the problem world context for CFDP. A typical problem world context for data transmission using the CFDP protocol includes a spacecraft, one or more ground stations, a network control centre and a third party, such as a laboratory or a remote user[1]. The remote user connects her/his laptop (containing a CFDP entity) to the NCC via the Internet, the NCC is then connected to a ground station which is itself connected to the spacecraft. This is an example of a "three parties" communication. In "two parties" communication, there is no third party: data is transferred between a spacecraft and a NCC. The protocol supports different qualities of service (reliable or unreliable)

[1] In principle, there can be more than one spacecraft and several remote users in some settings. The context of CFDP in Figure 1 is an illustrative, rather than definitive, example used in this paper.

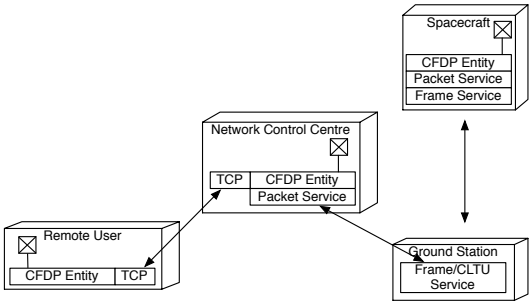

Figure 1. Problem World Context of CFDP [9]

and different kinds of links: up (from ground to spacecraft), down (from spacecraft to ground), or both.

In order to cater for customers with different CFDP needs, the developers have created a library of CFDP features as a software product line. Depending on the needs of customers, developers select various features into a product. In doing so, several constraints need to be satisfied:

- Stakeholder Requirements: The requirements relating to the behaviour of the system have to be satisfied by the product.
- Feature Dependencies: Some features are dependent on other features, and the product created needs to respect those dependencies.
- Contextual Constraints: In the field of on-board software, CPU usage and memory footprint are two quantities that have to be minimised, for reasons of the cost of the hardware. There should be no extraneous feature in feature configurations.

Currently, there is no formal way of systematically relating the feature configuration to the stakeholder requirements and the contextual constraints. Configuring large systems requires a high degree of knowledge about feature implementation and domain expertise, and when the configurator does not have access to the knowledge and expertise, producing appropriate configurations is time-consuming and error-prone. Therefore, configuring a product that meets the requirements for system behaviour as well as satisfies the constraints imposed by the hardware is a real challenge faced by the developers.

2.2 The Jackson-Zave Framework

Introduced in [35] and further expanded in [17], the Jackson-Zave framework is an important paradigm in requirements engineering. It has some key principles, two of which are relevant to our discussions in this paper. One

principle is concerned with the properties of software artifacts. Intuitively, it suggests that requirements (R) are expressed in terms of properties of the problem world context (W), and specifications (S) which, *within the problem world context*, are expected to satisfy the requirements. This relationship is often described using the logical entailment operator as: $S, W \vdash R$.

For example, the simplified requirement for a reliable uplink in a two party communication (R_{ru}) is to ensure that the file sent by a user at NCC arrives at the spacecraft with a high degree of success. The specification of this feature will describe the behaviour of the software at the NCC end and at the spacecraft end (S_{ru}). It has to rely on the structure and behaviour of the problem world context, including the fact that the NCC is connected to one or more ground stations, which are in turn connected to the spacecraft, and how the ground stations behave when a file is relayed (W_{ru}). The important point here is that the specification and the requirements are two separate descriptions that can be connected through the description of the problem world context in which the specification is deployed.

The second principle is related to the notion of "adequacy argument", which describes how the three descriptions are related and thus justifies that the logical entailment $S, W \vdash R$ holds. In essence, it describes the causality that binds the three descriptions. In the case of the reliable uplink feature, it shows how the requirement (R_{ru}) is satisfied by (S_{ru}) and (W_{ru}).

2.3 Feature Modelling

Our approach is non-prescriptive about the specific syntax used to describe the dependencies between features. We assume that an FM is a description of how features are related, typically involving the *and/or*, *requires/excludes*, and *cardinality* relationships. Graphically, an FM may be either a tree or a table or simple textual descriptions. For readability we use trees, specifically, a single-rooted DAG (Directed Acyclic Graph) as defined by Schobbens *et al.* [28]. Additionally, each FM may have constraints that cannot be easily expressed diagrammatically, which are written as in a logical language such as propositional logic [3]. We assume that such an FM can be translated into a propositional formula FM so that a feature configuration (i.e. a set of features) fc is a valid assignment of values to variables in the formula: $fc \models FM$ [3, 23].

3 Quantitative Constraints

In our approach, we follow existing suggestions to extend classical FMs with quantitative attributes [4, 26], but try to be more systematic and align them with the Jackson-Zave framework introduced before. We will first discuss the syntactic changes wrt. FMs introduced in Section 2.3 and then revise the semantics.

3.1 Extending the syntax

In the syntax, we distinguish three main concepts: qualitative attributes, statements and constraints.

3.1.1 Attribute declaration

A quantitative (or feature) attribute is a named and typed property of a feature, which can be measured. We may, for instance, consider attributes such as CPU time or memory usage of a feature. An attribute is defined on a per-feature basis and written as a syntactic annotation of a feature node.

```
<qattribute> ::= <type> <name>
```

Where `<type>` denotes a primitive datatype and `<name>` a unique identifier. Which datatypes can be used is dictated by the reasoning engine used to implement these q-constraints. For the purpose of this paper, we limit ourselves to integers.

Note that the assumption that a quantitative attribute, generally representing a software quality, can be measured is rather strong. Actually, there are two factors. Firstly, it is generally acknowledged that many of the classical software qualities are too subjective to be quantified [21]. This problem can be overcome by considereing technically well-defined properties, such as memory or CPU consumption. Secondly, it might be hard to specify these qualities on a per-feature basis. This limitation also depends on what a "feature" denotes, in our example case, memory consumption, for instance can be specified for each feature[2]. CPU consumption, however, is not strictly additive and requires a domain-specific calculus. To this end, *statements* are a flexible mechanism for defining and aggregating attribute values.

3.1.2 Statements

A feature attribute is only a declaration of a (generally numeric) variable. To define the values of the attributes, features have to be annotated with *statements*:

```
<qstatement> ::= "[" <guard> "]" <definition>
<guard> ::= "selected" | "deselected"
<definition> ::= <name> "=" <function>
```

The `<guard>` denotes whether the definition holds for the case when the feature is selected or deselected. For each feature, there should be two statements with different

[2]Provided that strict coding conventions are followed by programmers, as it is the case with our industry partner

guards. The <definition> is a function over the attributes of the feature's sub-features.

This system can either be used to assign each feature attribute a static value, or to define a hierarchical domain-specific calculus, where the value of the attribute of a parent feature is calculated as a function of the child feature's values.

In general, the definition of a parent feature attribute will be a function aggregating the values of its children, and the definition of a leaf feature attribute a static value. The definitions of 'deselected' feature attributes will then be constants neutral wrt this aggregation function. For example, if the parent feature is calculated by adding up the attribute values of its children, the definition for a 'deselected' child feature would be simply "0".

3.1.3 Constraints

Whereas statements describe the values that feature attributes take, *constraints* defined on these attributes allow to limit the variability of the FM. They are also feature annotations with the following syntax:

```
<qconstraint> ::= "[" <guard> "]" <expression>
```

The <guard> has the same meaning as before. The <expression> is a predicate over all attributes which will evaluate to true or false for given attribute values. As for attribute types, the syntax of these expressions depends largely on the reasoner used to automate q-constraints. Basically, they have two effects: they can be used to restrict the values of feature attributes, or to exclude feature combinations that violate certain rules. The choice of a given type of spacecraft could, for instance, limit the allowed memory usage of a configuration of the CFDP library.

Note, however, that most constraints ought to have a 'selected' guard; intuitively the non-selection of a certain spacecraft type should have no influence on the memory usage limit; this aspect would thus be captured by a single constraint with a 'selected' guard attached to the feature representing the spacecraft type.

3.2 Revising the satisfaction relation

In Section 2.3, we defined the basic satisfaction relation $\models_B$: a configuration fc is valid if its boolean encoding satisfies the boolean encoding of the feature digram FM:

$$fc \models_B FM.$$

Now that attributes and constraints have been added, a configuration of an FM consists of a set of features and a set of values for the feature attributes. For such a configuration to be valid, it needs to satisfy the constraints on its attribute values as well, i.e.

$$fc \models_{CSP} FM,$$

where $\models_{CSP}$ denotes that the Constraint Satisfaction Problem (CSP) encoding of fc satisfies the CSP encoding of the FM. These encodings are defined by Benavides et al. [4], whose algorithm covers most of the constructs identified in the previous section. Their algorithm translates an FM with annotations such as ours into a CSP [24]. The algorithm cannot be used as-is, as Benavides et al. do not make use of explicit guards (they assume "0" to be the default value in all cases). Indeed, a statement/constraint $stmt$ with a 'selected' guard attached to a feature f will become the CSP constraint $(f \Leftrightarrow stmt)$ whereas one with a 'deselected' guard translates to $(\neg f \Leftrightarrow stmt)$.

This leads us to define the general satisfaction relation.

Definition 1 *Let fc be a configuration and FM an FM then*

$$fc \models FM \quad \textbf{\textit{iff}} \quad fc \models_B FM \wedge fc \models_{CSP} FM.$$

3.3 Optimisation

During the configuration of an FM, constraints help to reduce the number of choices that have to be made. Additional use can be made of the feature attributes by specifying an optimisation goal [4].

The idea it to specify a function over the feature attributes, which has then to be optimised (maximised or minimised) given the CSP encoding of the diagram. This way one can obtain configurations that are optimal wrt an objective function. In the present case, memory and CPU usage will be restricted by constraints, but can in addition be minimised to obtain not only configuration within the limits, but truely minimal configurations. The objective is not part of the FM itself, since it depends on the requirements and as such is assumed to be added towards the end of the configuration process. For instance, if a configuration of the CFDP library is built for a network control centre (rather than a spacecraft), memory usage need not necessarily be minimised.

4 The Proposed Approach

The section discusses our approach, including the separation of FMs into FMs related to requirements, problem world contexts, specifications, expression of quantitative constraints, and a general procedure for obtaining an optimal configuration.

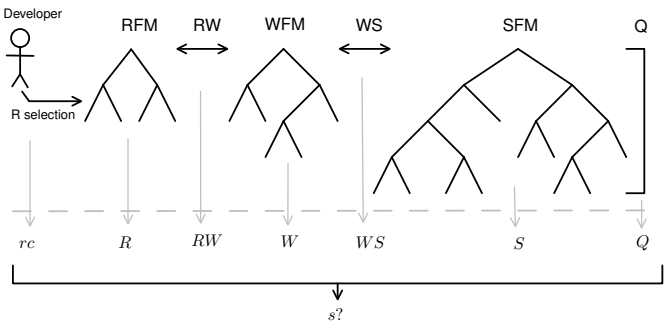

Figure 2. A Schematic overview of our approach

4.1 Overview

As shown in Figure 2, in this approach, feature descriptions are separated into three FMs relating to the requirements, problem world contexts and the specifications (RFM, WFM and SFM) respectively. These models are connected by two sets of cross links (xlinks) [25] over them (RW and WS). In addition, there may be quantitative constraints (Q-constraints) over each of the models.

Propositional formulae of RFM, WFM and SFM will be referred to as R, W and S, while rc denotes the configuration of R provided by the developer. Propositional formulae of links between the FMs will be referred to as RW and WS respectively. Formulae of Q-constraints will be referred to as Q.

The challenge is to find one or more optimal configuration of the feature specifications for the selected requirements, represented by s, if they exist.

4.2 Separating Descriptions of Feature Models

The three FMs of our approach are as follows:

1. Requirement FM (RFM): RFM describes different requirements that can be satisfied by the product line. This model is "high-level", in the sense of reflecting the requirements engineers view of the system, and the number of features in this model being fewer than other models. Therefore, the ability to generate optimal configurations largely from this model helps solve the problem of relating requirements to feature configurations.

2. Problem World FM (WFM): This model describes the features in the system context by showing different physical settings in which the software system might be deployed. This model reflects the view of system engineers who design the system hardware.

3. Specification FM (SFM): This model describes the feature of the software, reflecting software engineers' view of the system.

4.3 Linking RFM, WFM and SFM

In principle there are two sets of xlinks: between requirements feature and the problem world FM, and between the problem world feature and the specifications feature. The first set of links (RW) shows the problem world configuration required by the requirements, and the requirements satisfied by the problem world configurations. The second set of links (WS) shows similar relationships between problem world configurations and features.

4.4 Q-Constraints in RFM, WFM and SFM

The different elements related to Q-constraints presented in Section 3 can be attached to our 3 different FMs in order to obtain a more accurate configuration of SFM. Attribute declarations and statements, defining attributes types and values, are typically attached to features of SFM but can also be associated to RFM and WFM. Then, constraints on the quantitative attributes defined in the statements can be associated to RFM and WFM in order to, for example, restrict the amount of memory available for a configuration. Finally, an objective function can be added to RFM. Optimisation of the memory usage of a configuration of the CFDP library is an example of such an optimisation function defined on the memory load attributes of SFM.

4.5 Generating Optimal Configurations

In this approach there are three FMs connected by xlinks. The developer will select a configuration of the requirements, and sometimes also in the problem world, and this approach will automatically generate optimal configuration(s) of software features.

With an optimal configuration, we refer to any configuration of SFM that satisfies the requirements (a selected configuration RFM) within a particular context (a configuration of WFM implied by the configuration of RFM), and the quantitative constraints.

We now describe a general procedure for obtaining the optimal configurations. Recall that R, W and S denote respective propositional formulae describing the requirement, problem world and specification FMs. Let $[\![R]\!]$, $[\![W]\!]$ and $[\![S]\!]$ be respective sets of valid assignments.

First, R, W and S should be *independently satisfiable*, i.e. the sets $[\![R]\!]$, $[\![W]\!]$ and $[\![S]\!]$ should be non-empty. Additionally, R, W and S should be *collectively satisfiable*, i.e. there is at least one valid assignment for R, W, S, RW, WS, and Q.

Given a valid configuration of the requirements ($rc \in [\![R]\!]$), a valid set of configurations of the problem world context ($w \subseteq [\![W]\!]$) is one satisfying the constraints RW: for any $wc \in w$, the following holds

$$wc \models_B rc, RW, W$$

Again, given any $wc \in w$, a valid set of feature configurations ($s \subseteq [\![S]\!]$) is one satisfying WS and Q: for any $sc \in s$, the following holds

$$sc \models_B wc, WS, S$$

$$sc \models_{CSP} Q$$

If the set s is empty, then there is no possible configuration for requirements selected in rc. If there is only one member in sc, then the configuration is the optimal one. If there is more than one member in sc, then there are several optimal configurations, where additional constraints may be applied.

The feature configurations identified satisfy the requirements and the quantitative constraints, and therefore should be optimal. In order to ensure that this is really the case, the following sanity check can be performed on the configurations. The aim is to identify any unnecessary feature in the configurations. For each member $sc \in s$, let f be a feature in sc, and let δ be $sc \setminus \{f\}$. Then we need to show that

$$\delta \not\models_B rc, RW, W, WS, S$$

That is, removal of any feature from the configuration will lead to a failure to satisfy the requirements.

Notice that we regard the requirements and the quantitative constraints as equally important. Therefore, all configurations found satisfy them. In cases where either quantitative constraints or requirements are regarded as more important, then they may be evaluated in a particular order so that when no configuration is found, it is possible to identify conditions that are too strong.

In summary, the developer in this approach chooses a particular configuration of the requirements by selecting and deselecting features in the requirements FM. Once a configuration is chosen, the validity of the configuration within RFM is checked. If valid, all valid configurations of WFM that satisfy the constraints over the RFM and WFM are selected (if they exist). For each configuration WFM, SFM are further generated. Since constraints prevent selection of unnecessary features, the configuration generated will be optimal for the requirements chosen.

Linking requirements and specifications allows us to reason meaningfully about feature configurations. It is now possible to obtain feature configurations that satisfy functional requirement as well as quantitative constraints. In addition, it is possible to identify configurations of SFM never required by the configurations of RFM and WFM, a configuration of RFM that can never be satisfied by SFM.

5 Running Example

We now turn to the running example introduced in Section 2, and work through the configuration of the CFDP file delivery protocol developed by Spacebel, while providing further details of our approach. We also discuss how an off-the-shelf tool can be used to support our approach, whilst noting some of its limitations.

5.1 Expressing the RFM, WFM and SFM

In the CFDP example, features of the requirements FM, extracted from the CFDP documentations, are shown in Figure 3. The diagram shows that there are three main features in the requirements model: the sizes of files to be sent or received can be small, large or both, the number of parties involved may be either 2 or 3, and the quality of services (QoS) for uplinks and downlinks may be either reliable or unreliable.

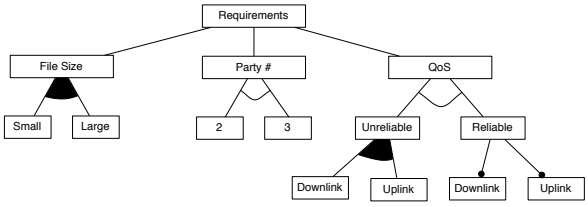

Figure 3. RFM

Features in the problem world context were obtained in the same way as ones in the requirements (see Figure 4). In every configuration, there has to be a spacecraft, NCC and at least one ground station. If the number of parties is 3, the third party has to be either a remote user or a laboratory, and if it is a laboratory, it may send, receive or do both. Moreover, there can be different types of spacecraft having different characteristics (*Type 1* and *Type 2*)

In the specifications of Spacebel, there are several features (see Figure 5): a terminal (such as spacecraft, NCC, remote user or laboratory) has to have at least one of the two features *recv_min* or *snd_min* in order to be able to receive or send files. Then, one can also select *pus* feature which is a protocol allowing remote control of a CFDP entity, *extended* to support file forward procedures and *reboot* to

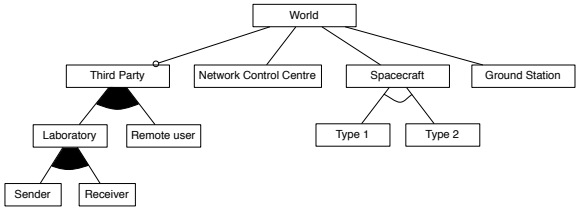

Figure 4. WFM

manage reboot of a CFDP entity in case of software crash. Those five features have themselves child features, but they are omitted for space reasons.

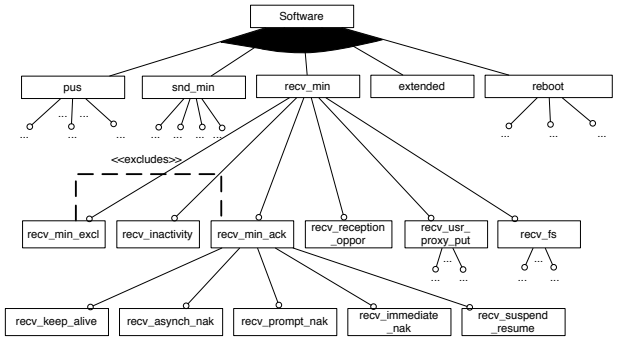

Figure 5. SFM

When translated into propositional formulae, nodes representing "significant features" in the models become labels, while other nodes become logical connectives.

5.2 Expressing constraints

Having described RFM, WFM and SFM of Spacebel, we can now express the constraints of the models according to our approach. The relationships between RFM and WFM are as follows:

$$3 \leftrightarrow Third\ Party \quad (1)$$

$$2 \leftrightarrow \neg\, Third\ Party \quad (2)$$

$$Remote\ User \rightarrow Reliable \quad (3)$$

$$Sender \rightarrow Uplink \quad (4)$$

$$Reicever \rightarrow Downlink \quad (5)$$

It says that if the option *3* (which indicates the number of parties) is selected in RFM, then *Third Party* has to be selected in WFM, and vice versa. It means that the world context has to include a third party if the user requires 3 parties. Similarly, if the option *2* is chosen, then *Third Party* has to be excluded from WFM. The problem world context requires that if *Remote User* is selected, then the requirement *Reliable* has to be selected too. Finally, if a laboratory is declared as *Receiver* or *Sender* in the world model then, features *Downlink* or *Uplink*, respectively, have to be selected in RFM.

We can also express the relationships between WFM and SFM as follows:

$$Spacecraft \rightarrow snd\_min \vee recv\_min \quad (6)$$

$$Network\ Control\ Center \rightarrow snd\_min \vee recv\_min \quad (7)$$

$$Sender \rightarrow snd\_min \quad (8)$$

$$Receiver \rightarrow recv\_min \quad (9)$$

The two first formulae are similar: they mean that if a CFDP entity, *Spacecraft* or *Network Control Center*, is selected that entity should at least be able to send or receives files, i.e. *snd_min* or *recv_min* should be selected in SFM. The two last formulae are also linked together: if a CFDP third party entity is declared as *Sender* (*Receiver*) in WFM then *snd_min* (*recv_min*) feature should be selected in SFM.

Finally, we will define the Q-constraints of our CFDP example. For the purpose of this illustration, we will consider a single quality only: memory consumption. The values and constraints presented here are chosen to illustrate. We first need to declare an attribute for each feature which we call *[featurename].mem*, of type integer. The statements for *recv_keep_alive.mem* are then the following:

$$[selected]\, recv\_keep\_alive.mem = 4$$

$$[deselected]\, recv\_keep\_alive.mem = 0$$

The first one means that if feature *recv_keep_alive* is selected, the value of its *mem* attribute will be *4*. Otherwise, its memory consumption will be *0*. The attribute statements of the other leaf features are similar.

For the non-leaf features, the statements express aggregation. For *recv_min*, this means:

$$[selected]\, recv\_min.mem = \Sigma_{f\ in\ children(recv\_min)} f.mem + 2$$

$$[deselected]\, recv\_min.mem = 0$$

where $children()$ returns the direct children of a feature. Again, the statements of the other non-leaf feature are similar. They mean that the memory usage of *recv_min* is the sum of the *mem* attributes of its sub-features plus 2 for overhead. The value of 'deselected' statements of the sub-features have to be defined in such a way that they will be neutral wrt the calculation of the memory usage of

recv_min. All statements are not represented here for reasons of conciseness.

Whereas SFM declares attributes and specifies their values, WFM will contain constraints over these attributes. Namely, we will attach two constraints to the subfeatures of the Spacecraft feature.

$$[selected]\, Software.mem \leq 15 \quad for\ \ Type\, 1 \quad (10)$$
$$[selected]\, Software.mem \leq 25 \quad for\ \ Type\, 2 \quad (11)$$

Those constraints mean that if a spacecraft of *Type 1* is selected in WFM, the memory load of a configuration of the CFDP library (represented in the *mem* attribute of the root of SFM) must be smaller than 15. For Type 2, the limit is 25. Those constraints, applied depending on the type of spacecraft selected, will rule out any product that have a memory load superior to the respective limits.

In this example, we have no constraints to attach to requirements. However, we can consider the following optimisation goal:

$$min(Software.mem), \quad (12)$$

meaning that the chosen configuration should be the one with the minimal memory load. Note that the goal depends on the requirements: if the configuration is built for a network control centre (with no memory limitation) the goal would not need to be considered.

5.3 From RFM, WFM and SFM to Feature Configurations

Having described the models and the constraints, we selected *Large* file size, a number of parties equal to *3* and a *Reliable* quality of service in RFM. The choice to have only 3 parties implies that one has to choose at least one of the two third parties in WFM (*Remote User* and *Laboratory*). We selected a *Remote User* and a *Type 1* spacecraft but SFM is not completely configured as one has to choose at least one of the two features *snd_min* or *recv_min*. Q-constraints should help to reduce the set of configurations by discarding those with a memory usage higher than 15. Morever, the objective function defined in RFM will give an optimal configuration.

5.4 Pure::Variants

The off-the-shelf tool we used is pure::variants, an Eclipse plug-in developed by pure-systems GmbH [5]. We chose pure::variants because of its maturity, stability and support for several aspects of our approach. The core elements of the software are the different models used to represent the problem domain, the solution domain and to specify a specific member of the family. Those different models are respectively named feature, family and variant description models in the tool.

The use of pure::variants in our Spacebel example can be decomposed in three main steps. First, we have to represent the three feature diagrams (RFM, WFM and SFM) using the syntax of the tool. This step includes the definition of the constraints linking elements of the same model. The *¡¡excludes¿¿* relationship between features *recv_min_excl* and *recv_min_ack* of SFM (Figure 5) is an example of such a relationship. The result of this first step for WFM is shown in Figure 6. Exclamation marks in front of features *Spacecraft*, *Network Control Centre* and *Ground Station* indicate that they are mandatory while the question mark associated with *Third Party* indicates that it is optional. Crosses in front of *Remote User*, *Laboratory*, *Sender* and *Receiver* indicate or-decompositons. The xor-decomposition, shown in Figure 6, is represented by an "equivalence" symbol. Note that all those marks in front of features refer to their parent feature and not child ones.

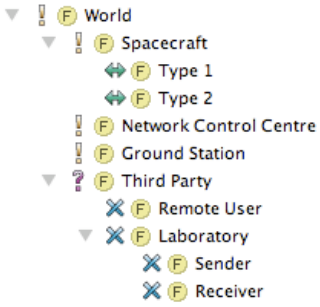

Figure 6. WFM in pure::variants

The advantage of pure::variants is that relationships between features of different feature models can be defined. This is our second step. The fact that a *Third party* in the WFM is incompatible with a *Party #* equal to 2 in the RFM is an example of such a relationship. Figure 7 contains that dependency as well as all others linking WFM with RFM and SFM presented in Section 5.1.

Once the constraints linking the features of the different feature models have been defined, a specific member of the family can be defined with variant description models. This is the final step. Figure 8 is the variant description model of our Spacebel example including the 3 feature models. There, in the *Requirements* model, we select a *Large* file size, a party number equal to *3* and an unreliable uplink transfer. Consequently, as *Remote User* requires reliable transfers, a third party *Laboratory* is automatically selected in the *World* model. Then we manually select *Type 1* spacecraft and a *Sender* role for the laboratory. The *Type 2* spacecraft is automatically discarded in the *World* model while *snd_min* is automatically selected in the *Software* model.

Finally, we also have to select a *File Size*. The result of

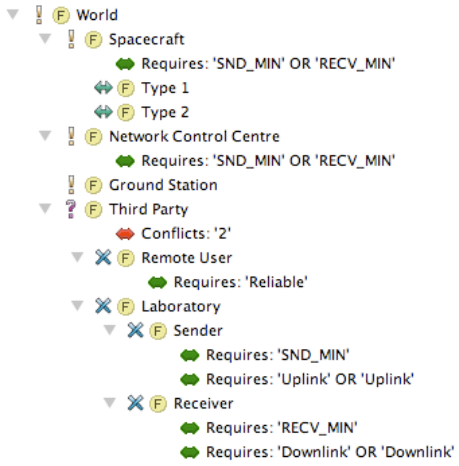

Figure 7. Dependencies of WFM in pure::variants

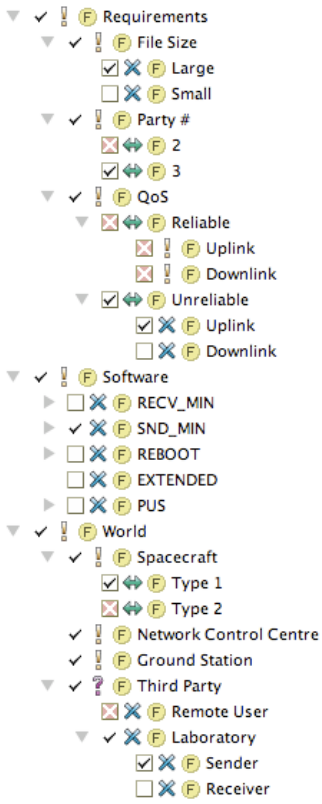

Figure 8. Variant Description Model of the Spacebel example

the choices made in *World* and *Requirements* is the automatic selection of *snd_min* feature in the *Software* model. It means that in order to satisfy constraints defined in WFM and RFM, the *snd_min* feature of SFM has to be present in the final product.

5.5 Limitations

Although we were able to implement a part of our approach in pure::variants, there are some important limitations, which are now discussed.

The first limitation was the impossibility to select several instances of the same feature in pure::variants. This option could be useful if we had a constraint like:

$$Large \quad \rightarrow \quad \#GroundStation > 1 \qquad (13)$$

It means that if large files have to be transferred, there should be several ground stations involved in the transfer.

The impossibility to easily check quantitative constraints over attributes was another weakness of the tool. We were able to calculate, for example, the sum of attributes of selected features and show it in the graphical user interface but it was not possible to automatically discard solutions wrt constraints and optimisation goals defined in WVM and RVM. Consequently, the part of our approach related to Q-constraints could not be fully implemented in pure::variants.

6 Related Work

In this paper, we chose to build on the extensive work carried out by Jackson, Zave and others in an attempt to clarify the notions "requirement" R, "specification" S and "context" W [36, 15, 17]. Jureta et al. [19] recently extended this work; they introduce the concepts of "quality constraint" Q and "attitude" A and change the monotonic entailment relation of the adequacy argument (shown in Section 2.2) to a non-monotonic one. This *core ontology* allows to structure the modelling/description space and makes relationships between different types of descriptions explicit. Our goal was to propose a comprehensive approach that completely covers this ontology, and that preserves the distinction of the different types of descriptions. This is, to the best of our knowledge, the first approach to do so, as shown in the following survey of existing approaches.

Requirements Engineering Approaches. The goal-oriented approach KAOS refines high-level goals into subgoals and then into operational requirements [31]. Operational requirements are then assigned to agents in the solution space [22]. Using a similar notion of goals and goal-refinement, the NFR framework discusses how goals may contribute to achieving software quality [6]. The i^* approach to RE attempts to augment specifications with contextual information such as the business model [32, 1]. They

all deal, however, with *single system development*, rather than SPLE and do not cover the W or the S (the latter except for KAOS and i^*).

Previous work by the authors. This work is also an extension of earlier work by some of the authors. In [7], Classen et al. suggested using the Problem Frames approach, derivative of the above Jackson-Zave framework, to structure and identify problem variability. This rather exploratory work was refined in [8] with a generic definition of "feature" as a tuple of R, W and S. Based on this definition, the satisfiability criterion for an FM was revised, taking into account the Zave and Jackson adequacy argument [36]. However, this paper failed to explicitly consider variability in W and S, assuming them to be part of a feature; that is variability in R, W and S was considered, but all in one dimension. A first move towards a multi-dimensional view was undertaken in [25], where Metzger et al. identify the necessity, and propose an approach for, distinguishing variability in S from variability in R.

Partial consideration of dimensions. Other authors have already proposed to consider multi-dimensional FMs. Kang et al. [20], as part of the FORM approach, define four *layers*, each containing a number of FDs. The paper, however, discusses these layers and their FMs on a rather informal level, and it is not made clear what links between features of different levels mean. Thiel and Hein [30] capture context variability and software variability in the same diagram. Yet, they never explicitly attribute features to context or software and do not discuss this aspect. Similarly, Hartmann and Trew [16] combine context and requirements in the same diagram. The context, in their case, however, is closer to the requirements than to the software, and basically serves to eliminate variability in the requirements. Desmet et al. [12] capture context and software variability in the same FM. However, as opposed to our and the previously mentioned approaches, they do not explicitly model variability in the context. They rather annotate decomposition links with context rules that merely decide when a feature should be activated or not.

Orthogonal and multi-level models. Batory [2] uses several *orthogonal* FMs for representing different aspects of the requirements. These FMs are, however, assumed to be unrelated. Reiser and Weber [27] propose to use multi-level feature trees; their purpose is to cope with large diagrams and distributed teams, rather than different concerns. Somewhat related is also the approach of multi-level staged configuration [11], where multiple FMs are used to model decisions that need to be taken by different people or at different points of the development cycle. Our work is complementary in that we provide a systematic way of separating the FMs and reason about the relationships between requirements and feature configurations.

Approaches to quantitative constraints in SPLE. Quantitative constraints were already part of earlier variability notations, such as the COVAMOF framework [29, 13]. In COVAMOF, constraints on quality attributes are expressed using *dependencies*, i.e. restrictions on variant selection, but so are also requirements and even $excludes$ links between variants, which leads to a situation where different kind of information (R, S and Q) is mixed. As mentioned before, goal modelling approaches explicitly deal with non-functional requirements, generally represented through *soft goals*. In this context, Yu et al. [34, 33] propose an approach for translating goal models to FMs, where contribution to softgoals is translated to guarded requires and excludes links in the FM. Jarzabek et al. [18] model quality attributes in a goal model which is linked to features of a set of orthogonal FMs that represent both S and W. Their approach is, however, not formalised or tool supported. Baixauli et al. [14] analyse variability at the level of the goal model only as a way to identify optimal alternatives wrt. to contribution links. However, they only consider goals, rather than features or domain concepts, making their approach suitable for early requirements engineering, rather than configuration as we do here.

In conclusion, existing work has looked at the problem of multi-dimensional FMs from various angles. In this paper, we have provided a systematic way of separating and linking FMs in order to reason about relationships between requirements and feature configurations.

7 Conclusions and Future Work

When configuring software features at the implementation level, it is often difficult to ascertain whether a given configuration is optimal with respect to stakeholder requirements and contextual constraints, and whether all requirements have been met by the configuration. In this paper we presented a systematic approach to relating requirements to feature configurations. Based on the Jackson-Zave framework, the proposed approach separates feature descriptions into FMs relating to the requirements, the problem world context and the specifications. Quantitative constraints such as the limitations of the CPU and memory costs of configurations are also taken into account.

As a result, by simply selecting requirements for a desired product, one or more feature configurations that satisfy the requirements and the quantitative constraints are generated. The advantages of our approach are twofold. First, the developer can work at the high level of requirements where the number of features is much lower than the feature in the specifications. Secondly, perhaps more importantly, because we take into the account the requirements and the context it entails, the feature configuration obtained is always optimal. Our approach is illustrated with an application to a real-life example of a satellite communication

software, and off-the-shelf tool to support our approach, and its limitations, were also discussed.

In future, we intend to extend this work in several ways. First, we intend to evaluate our approach on larger case studies, including those from Spacebel. Second, we intend to provide better tool-support for describing and evaluating quantitative constraints. Finally, we intend to explore how our approach can be synthesised with the multi-stage configuration framework.

8 Acknowledgements

We thank Spacebel, in particular, Arnaud Bourdoux and Laurent Demonceau, for pointing us to the CFDP example and providing validating comments on our work. We also thank our colleague Anthony Clève for his comments on the paper.

References

[1] F. M. R. Alencar, J. Castro, G. A. C. Filho, and J. Mylopoulos. From early requirements modeled by the i* technique to later requirements modeled in precise uml. In *WER*, pages 92–108, 2000.

[2] D. Batory, J. Liu, and J. N. Sarvela. Refinements and multi-dimensional separation of concerns. *SIGSOFT Softw. Eng. Notes*, 28(5):48–57, 2003.

[3] D. S. Batory. Feature Models, Grammars, and Propositional Formulas. In *SPLC'05*, pages 7–20, 2005.

[4] D. Benavides, P. T. Martín-Arroyo, and A. R. Cortés. Automated reasoning on feature models. In *CAiSE'05*, pages 491–503, Porto, Portugal, June 2005.

[5] D. Beuche. Modeling and building software product lines with pure: :variants. In *SPLC'08*, page 358, Washington, DC, USA, 2008. IEEE CS.

[6] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.

[7] A. Classen, P. Heymans, R. Laney, B. Nuseibeh, and T. T. Tun. On the structure of problem variability: From feature diagrams to problem frames. In *VaMoS'07*, pages 109–117, Limerick, Ireland, January 2007.

[8] A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a feature: A requirements engineering perspective. In *FASE 08, held as Part of ETAPS'08*, pages 16–30. Springer, 2008.

[9] Consultative Committee for Space Data Systems (CCSDS). *CCSDS File Delivery Protocol (CFDP): Blue Book, Issue 4 and Green Book, Issue 3*. Number CCSDS 727.0-B-4, CCSDS 720.1-G-3. NASA, 2007.

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

[11] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[12] B. Desmet, J. Vallejos, P. Costanza, W. D. Meuter, and T. D'Hondt. Context-oriented domain analysis. In *CONTEXT 2007*, pages 178–191, 2007.

[13] L. Etxeberria, G. Sagardui, and L. Belategi. Modelling variation in quality attributes. In *VaMoS'07*, pages 51–59, Limerick, Ireland, January 2007. LERO.

[14] B. Gonzalez-Baixauli, J. C. S. d. P. Leite, and J. Mylopoulos. Visual variability analysis for goal models. In *RE'04*, pages 198–207, Washington, DC, USA, 2004. IEEE CS.

[15] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, 2000.

[16] H. Hartmann and T. Trew. Using feature diagrams with context variability to model multiple product lines for software supply chains. In *SPLC'08*, pages 12–21, Washington, DC, USA, 2008. IEEE CS.

[17] M. Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison Wesley, 2001.

[18] S. Jarzabek, B. Yang, and S. Yoeun. Addressing quality attributes in domain analysis for product lines. *Software, IEE Proceedings*, 153(2):61–73, April 2006.

[19] I. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *RE'08*, pages 71–80, 2008.

[20] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Software Eng.*, 5:143–168, 1998.

[21] B. Kitchenham and S. L. Pfleeger. Software quality: The elusive target. *IEEE Software*, 13(1):12–21, 1996.

[22] E. Letier and A. van Lamsweerde. Agent-based tactics for goal-oriented requirements elaboration. In *ICSE'02*, pages 83–93, New York, NY, USA, 2002. ACM Press.

[23] M. Mannion. Using First-Order Logic for Product Line Model Validation. In *SPLC'02*, pages 176–187, San Diego, CA, Aug. 2002.

[24] K. Marriott and P. J. Stuckey. *Programming with Constraints : An Introduction*. MIT Press, 1998.

[25] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *RE'07*, pages 243–253, New Delhi, India, October 2007.

[26] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, NJ, USA, 2005.

[27] M.-O. Reiser and M. Weber. Managing highly complex product families with multi-level feature trees. In *RE'06*, pages 146–155, Los Alamitos, CA, USA, 2006. IEEE CS.

[28] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *RE'06*, pages 139–148, Minneapolis, Minnesota, USA, September 2006.

[29] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. In *SPLC'04*, pages 197–213, 2004.

[30] S. Thiel and A. Hein. Modeling and using product line variability in automotive systems. *IEEE Software*, 19(4):66–72, 2002.

[31] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of Fifth IEEE International Symposium on Requirements Engineering, 2001*, pages 249–262, 2001.

[32] E. S. K. Yu and J. Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 159–168, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[33] Y. Yu, J. C. S. do Prado Leite, A. Lapouchnian, and J. Mylopoulos. Configuring features with stakeholder goals. In *SAC'08*, pages 645–649, New York, NY, USA, 2008. ACM.

[34] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. C. S. P. Leite. From goals to high-variability software design. In *ISMIS 2008*, pages 1–16, 2008.

[35] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM TOSEM*, 6(1):1–30, 1997.

[36] P. Zave and M. Jackson. Telecommunications service requirements: Principles for managing complexity. 1997.