MASTER EN SCIENCES DE GESTION

La culture organisationnelle: entre volonté d'intégration et stratégies de mise à distance: le cas de Boehringer Ingelheim

Baudoux, Céline

Award date:
2009

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'Informatique
Année académique 2008-2009

Automated Software Configuration: the Product Line Approach

An Application to Conference Management Websites

Baudoux Critophe

Mémoire présenté en vue de l'obtention du grade de Maître en Sciences Informatiques

Acknowledgements

I would particularly thank:

Mr Patrick Heymans, my supervisor and professor at the Faculty of Computer Science at the University of Namur, to have directed me throughout the elaboration of this master thesis ;

Mr Arnaud Hubaux and Mr Quentin Boucher for the time they have spent to guide me and to answer to my questions, as well as for the frequent rereading ;

Mr David Benavides Cuevas, professor at the University of Seville, to have integrated me as trainee in the Department of Computer Languages and Systems, and to have followed, advised, helped me throughout my traineeship ;

The whole staff of the Department of Computer Languages and Systems for its hospitality and good mood ;

And finally, my parents and friends to have supported me unconditionally during the elaboration of this work.

Abstract

The active researchers and engineers in diversified domains need to communicate, to archive the obtained results and to state their advances. At this end, they publish articles in specialist magazines or participate at scientific conferences, which take generally place annually.

The organization of these last ones passes in particular by the creation of a website presenting the practical modalities. However, the organizers have generally not enough time to dedicate to this task. The conferences have a well defined organization and have recurring operating modes. This last characteristic is typical of software product line. Thanks to this approach, we can envisage the automation of the creation of a product, namely, in our case, a website of conference.

In this master thesis, we present two approaches allowing to automate the creation of a website of conference to lighten the time dedicated to this task. These two approaches have totally opposite optics: the first one proposes a graphical interface allowing to guide users in their choices and the second one, more technical, allows to select the characteristics directly on a feature diagram. On basis of the collected information, they generate automatically the website of conference. Beyond these two solutions, our contribution will be also supported by the comparison of these two methods in order to identify their strengths and weaknesses. On basis of the results, we shall present a unified approach.

Résumé

Les chercheurs et ingénieurs actifs dans divers domaines ont besoin de communiquer, d'archiver les résultats obtenus et faire état de leurs avancées. A cette fin, ils publient des articles dans des revues spécialisées ou participent à des conférences scientifiques, qui ont lieu annuellement.

L'organisation de ces dernières passe notamment par la création d'un site internet présentant les modalités pratiques. Cependant, les organisateurs n'ont généralement que peu de temps à consacrer à cette tâche. Les conférences ont une organisation bien établie et sont dotées de modes opératoires récurrents. Cette dernière caractéristique est typique des lignes de produits logiciels. Grâce à cette approche, on peut envisager l'automatisation de la création d'un produit, à savoir, dans notre cas, un site internet de conférence.

Dans ce mémoire, nous présentons deux approches permettant d'automatiser la création d'un site web de conférence afin d'alléger le temps consacré à cette tâche. Ces deux approches ont des optiques totalement opposées : l'une propose une interface graphique permettant de guider l'utilisateur dans ses choix et l'autre plus technique permet de sélectionner les caractéristiques directement sur un feature diagram. Sur bases des informations recueillies, ils génèrent automatiquement le site internet de conférence. Au-delà de ces deux solutions, notre contribution sera également appuyée par la comparaison de ces deux méthodes afin d'identifier leurs forces et faiblesses. Sur base des résultats, nous présenterons une approche unifiée.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since many years, Software Product Lines (SPLs) have been presented as a promising approach to improve software quality and productivity. SPLs have been used by the manufacturing industry for a long time to reduce costs and increase productivity for software development [61]. According to customer requirements, a product configuration allows to derive a list of all standard features and available options for each product.

In this chapter we present the context of this master thesis with an introduction to SPL and feature-based product configuration. Then, we define the problem to solve and the different strategies which could be used to solve it. The structure of the thesis is finally defined at the end of the chapter.

1.1 Context

SPLs refer to software engineering methods, tools and techniques for creating a collection of similar software systems from a shared collection of software assets using a common way of production [78]. SPL is an important software development paradigm allowing to yield enormous gains in quality, costs, time to market, productivity and carry out order-of-magnitude improvements in other business drivers. "Manufacturers have long employed analogous engineering techniques to create a product line of similar products using a common factory that assembles and configures parts designed to be reused across the varying products in the product line" [61]. For instance, automotive manufacturers can produce unique variations of one car model using a set of designed pieces and a platform especially designed to assemble and configure those pieces.

SPLs are distinct from previous works by pro-active against opportunistic

software reuse. In a well defined SPL, when reuse is predicted in one or more products, software artefacts are called to create different products of the SPL rather than put general software components into a library with the hope that reuse will be possible. Recent advances in the SPL domain have showed that software engineering capability can be improved thanks to a narrow and strategic application of these concepts [66]. SPL engineering has a competitive business advantage in providing rapid market entry and flexible response, and allowing mass production and mass customization. The mass production represents a big advance in the manufacturing domain and can be defined as the ability to efficiently produce many copies of the same product, which is trivial. Advance in software engineering and manufacturing, the mass customization can be defined as the ability to efficiently produce many variations of a single product. Commonality and effective management of the variation (variability) in a product line are key elements of mass customization.

Many SPLs use product configuration techniques for managing the variability between their applications. Product configuration systems allow to configure modular products in function of the customer requirements. Such systems list all mandatory features as well as available options for each product. Configuration is the process of deriving a concrete configuration conforming to a Feature Diagram (FD) by selecting and cloning features, and specifying attribute values [64]. FDs are a well accepted means for expressing requirements in a domain on an abstract level. They are applied to describe variable and common properties of products in a SPL, and to derive and validate configurations of software systems [82]. FDs are visually represented as a tree where primitive features are leaves and compound features are interior nodes. One of the key success elements in SPLs is to achieve reusability, adaptability, and configurability of SPL assets like architectures and features. These goals are achieved by the identification of commonality and variability properties among products of a product line. Feature modelling is an important approach for capturing commonalities and variabilities in SPL [65].

1.2 Problem to solve

An important problem to solve is the automation of the creation of the core assets available in the SPL. As we have seen above, configuration is essential in the optic of SPL because it pilots product configuration according to customer requirements. In this master thesis, we illustrate this problem on conference website product lines. The automation of the creation of conference website has some advantages. First, it allows to simplify the process which can be complex and thus, to spare some precious time that people generally don't

have. Second, users do not require new technical knowledge (*HTML*, *PHP*, etc.) to build the website and they do not need a professional to make the task, allowing them to save money.

That problem can be subdivided into three sub-problems. The first one is the translation of the conference website product line into a particular conference website ready to use. Different techniques to start from the SPL and go to the final result should be developed. The second one is a problem of representation of the SPL to be comprehensible for users. They must have a global view of the features of the SPL to configure a product. The third problem is the validity of a product composed by users: verifications should be performed in order to avoid errors. We thus need to find a means to validate products. We have to find strategies to solve these problems.

1.3 Solution strategies

The contribution of this work is to automate the creation of a product using the SPL approach. In order to reach that goal and solve the problems introduced in the previous section, two strategies will be analysed.

The first strategy is an application specific approach which consists in the development of a wizard generating a conference website. This wizard is user-friendly and easy to use. It allows users to select features they want in the website. At each step of the configuration, they will be helped in their choices.

The second strategy is a generic approach which uses variability modelling and configuration tools to generate the conference website. Those tools require more experience from the user. They take one or more models (representing the product line) as input and allow users to configure a product. The different tools will be compared in order to highlight their strengths and weaknesses.

1.4 Outline

This master thesis is divided into four chapters:

In **Chapter 2**, the background of our work is presented and the general principles used in this master thesis such as SPL, feature diagrams (FDs), and variability modelling and configuration tools are introduced.

In **Chapter 3**, we introduce the running example: the conference website product line. We begin to introduce the principles of conference management and Content Management Systems (CMSs). Then we carry out a requirements engineering of the system to build followed by the definition of the conference product line. Finally we make the selection of a CMS.

In **Chapter 4**, we develop two different approaches to automate the creation of a conference website. We first explain an application specific approach and then a generic one which can be used with different tools. We also propose a structured comparison of those approaches.

In **Chapter 5**, a unified solution which combines pros of both approaches presented in the Chapter 4 is proposed.

Finally we present contributions, limitations and perspectives of this work in the concluding chapter.

Chapter 2

Background

This chapter presents the necessary concepts for the understanding of our solutions. First, we define what a SPL is. Second, we describe the FDs and finally we present a selection of variability modelling and configuration tools for SPL.

2.1 Software product lines

During those last years, the manner to produce goods has radically changed. Before, artisans produced goods for individual clients. As one goes along years, we have seen an increase of the number of people able to buy different kinds of products. For instance, in the automobile area, this led to production lines allowing for a mass market to produce at lesser costs than hand-made individual products. However, the number of possible diversifications has decreased because of production line.

We can classify products, those individual and those issued of *mass production* in the software area as: standard and individual softwares [78]. But both types of software products have their disadvantages, the standard software products have a lack of adequate diversification whereas the individual software products are expensiver. Standardized mass products satisfied the most of customers but, to stay in the automobile area, some of them would not use the same type of car for any goals. Indeed, some cars are used for making races, others for going to the work place. Others are made to welcome two persons whereas certain welcome six persons. This implies a rising demand for individualized products that confront the car industry. "This was the beginning of *mass customization*, which means taking into account the customers' requirements and giving them what they want" [78]. In order

to reach the customer wishes in term of customization, SPL engineering encourages the production of software products with common features instead of producing them one by one [53]. Mass customization can be defined as " the production of goods and services to meet individual customer's needs with near mass production efficiency" [87]. In this definition, two concepts are represented, the first is that mass customization meets individual customer's needs whereas the second is that the first is possible thanks to the mass production efficiency. The production of individual products of an organization producing mass production goods can be substituted by the production of similar products allowing flexibility because they share a common part. The mass production aims to produce a large quantity of standardized products by the use of standardized processes. These processes allow to the same product to be produced a lot of times in a reduced time to market. In mass production, there is no customization of products because the customers' requirements are identical.

"A SPL consists of a set of software products sharing a common set of features that satisfy the needs of a particular domain and that are developed from a common set of core assets in a prescribed way" [61]. SPL is an important software development paradigm allowing to yield enormous gains in quality, cost, time to market, productivity and accomplish order-of-magnitude improvements in other business drivers. For that, SPL engineering promotes the development of standard systems rather than individuals. In a same domain, a set of software products can vary from one to another, this is the variability between products which is captured by SPLs. "The variability designates elements that may vary from a product to another one" [94]. SPL engineering is divided in two complementary activities: *domain engineering* and *application engineering*. The purpose of the *domain engineering* is to produce software assets which are employed in different products of the SPL. It is also named core asset development. *Application engineering*, also called product development, aims to produce individual systems on basis of individual needs and core assets. In this section, we begin by defining the motivations for using SPL. Then, we explain the concept of variability and we finish with the description of the SPL engineering framework.

2.1.1 Motivations for SPL engineering

The main goal that SPL engineering searches to reach is to provide customized products for minimum costs. Here, we briefly describe the motivations to use SPL engineering for the software development.

Reduction of developments costs The costs reduction is an important pretext to introduce SPL engineering. The reuse of artefacts issued of a platform in different types of systems allows to considerably reduce the costs for each system. However, we have to make indispensable investments in order to reuse artefacts. Before reducing the costs, investments for the creation of the platform are necessary.

Quality enhancement As artefacts in the platform are reused in a lot of products, they are frequently tested and possibly improved. These improvements allow each time to increase the general quality of the products by detecting and correcting errors.

Reduction of time to market The SPL engineering allows to reduce the time to market because of the reuse of artefacts. In SPL engineering, only the first development of artefact takes time as for the development of each individual product. Once the artefact is developed, it can be reused for each new product and thus reduce the time to market.

2.1.2 Variability

SPLs can have several forms: in some ways, every product in a SPL use the same architecture without fit it, whereas in other ways, the architecture of the different products may vary [58]. In the same way, in some SPLs, only one configurable component implementation can be related to each architectural component, whereas, multiple component implementations are possible for an architectural component in other SPLs. Those different forms describe the variations between the products by taking advantage of different variability mechanisms.

"Software variability is the ability of a software system or artefact to be changed, customized or configured for use in a particular context. A high degree of variability allows the use of software in a broader range of contexts, i.e. the software is more reusable" [57].

In SPLs, the variability is expressed through variation points. "Variation points represent unbound options about how the software will behave" [71]. For each variation point in the production process, product decisions are used to opt for some options and so entirely specify the variation point behaviour in the final product. The moment at which the decisions are bound for a variation point is often called binding time. In a SPL, multiple binding times may be used allowing to bound some decisions earlier in the lifecycle and defer others later in the process. Among the different binding times, there

are the following: source reuse time, development time, language design time, program writing time, compile time, load time, run time, etc.

2.1.3 SPL engineering framework

The SPL engineering framework is divided into two key processes: *domain engineering* and *application engineering*. Figure 2.1 shows the *domain* and *application engineering* processes which are described in the following subsections.

Domain engineering

"The domain engineering process is responsible for defining the commonality and the variability of the product line, and thus for establishing the reusable artefacts" [78]. To produce applications that are within the scope of the SPL, the role of the domain engineering is to ensure that the appropriate variability is available. That implies to provide mechanisms for performing variability in the respective development artefacts, for instance by designing configurable components.

The domain engineering process includes four sub-processes such as *domain requirements engineering*, *domain design*, *domain realization* and *domain testing* (as shown in upper part of Figure 2.1 issued of [78]).

Domain requirements engineering The domain requirements engineering sub-process includes the documentation and elicitation activities to highlight the common and variable requirements of the SPL. This sub-process receives in input the product roadmap. Its output is reusable, textual and model-based requirements and, more particularly, the variability model of the SPL. The output contains the common and variable requirements for all possible applications of the SPL and not those of a particular application.

Domain design This sub-process includes the definition activities of the SPL reference architecture. This architecture determines the high level and common structure for every SPL applications. The input of this sub-process is composed of the domain requirements and the variability model issued of the previous process, the domain requirements engineering. The output provided is a refined variability model and the reference architecture. The refined variability model contains the necessary variability for technical reasons and named internal variability.

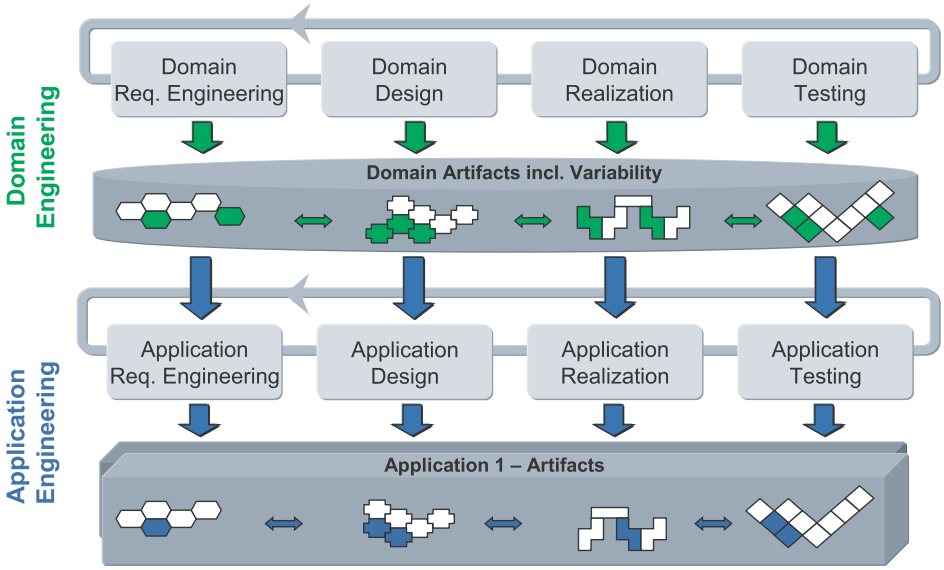


Figure 2.1: The software product line engineering framework

Domain realization The detailed design and the implementation of reusable software components are accomplished in the domain realization sub-process. The input is composed of the variability model and the reference architecture which encompasses a set of reusable software artefacts to elaborate in this sub-process. The output is the detailed design and implementation assets of reusable software components.

Domain testing The domain testing sub-process deals with checking and validating the reusable components. The components are assessed in function of their specifications such as their architecture, requirements and design artefacts. Domain testing provides also, to reduce the application testing effort, reusable test artefacts. Its input is composed of different elements: implemented reusable software components, components and interfaces designs (issued of domain realization), the reference architecture (issued of domain design) and domain requirements (from domain requirements engineering). The output includes reusable test artefacts and test results executed in this sub-process.

Application engineering

The application engineering process deals with the derivation of SPL applications from reusable artefacts. "Application engineering exploits the variability of the reusable artefacts by binding the variability according to application specific needs" [78].

In the application engineering process, four sub-processes are included such as *application requirements engineering*, *application design*, *application realization* and *application testing* (as shown in lower part of Figure 2.1 issued of [78]).

Application requirements engineering The application requirements engineering sub-process includes development activities for application requirements specification. The quantity of domain artefact reuse that can be completed is dependent of the application requirements. Consequently, identify the platform available abilities and detect deltas between applications requirements are the main concern of the application requirements engineering. The input includes the domain requirements and the product roadmap completed with the main characteristics of the corresponding applications. It can be completed in addition by specific requirements for the particular application, those that have not been identified during the process of domain requirements engineering. The requirements engineering specification for the particular application composes the output.

Application design This sub-process includes the production activities of application architecture. The application architecture is instantiated by the reference architecture defined in this sub-process. It also makes the selection and the configuration of the required parts of the reference architecture. Application-specific adaptations are also added. The variability bound is linked to the complete structure of the system. The input of the application design is the application requirements specification and the reference architecture issued of the previous sub-process, whereas the output is composed of the application architecture for the application.

Application realization The application realization sub-process implements the application. The major activities are to select and configure the reusable software components, and complete the application-specific assets. Reusable software components and application-specific assets compose the application.

The input of this sub-process is composed of the application architecture and the reusable realization artefacts issued of the platform. The output is a running application and the design artefacts.

Application testing This sub-process consists of the activities dealing with the validation and verification of the application in relation to its specification. The input of this sub-process is composed of a test reference which uses the application artefacts, the implemented application, and the reusable test artefacts issued of the domain testing. The output contains a test report consisting of the test results and problem reports with the detected failures of the application.

2.2 Feature diagrams

"SPL Engineering (SPLE) is an emerging software engineering paradigm, which guides organizations toward the development of products from core assets rather than the development of products one by one from scratch" [73]. The reuse of existing assets is the basis of the creation of a product issued of a SPL. For a specified SPL, some of these assets are particular for individual products whereas others are common to other products [53]. When coping with SPLs, the software engineering is confronted to the manner to express a SPL to highlight the commonality and the variability between products and express a specific product. FDs are a widespread means to represent SPLs. A feature is so a particular product and can be assimilated to an increment in the product functionality [69]. "Feature modelling is one of the most popular domain analysis techniques" [73] allowing to build highly reusable core assets for a SPL by the analysis of commonality and variability in the domain.

In a SPL, all the possible products are represented by a FD in a unique model in which the commonality and variability of the SPL are captured. Each relevant property is defined as a feature in the model. The commonalities and variabilities are described in a feature, which is an abstract concept meaning the needs for each SPL to be decided. In this meaning, a feature is, for some stakeholders, a relevant property of a system. It can be a requirement, a technical function or function group or a non-functional (quality) property, etc. determined by the stakeholders interest [56].

A FD is a set of features having a precise hierarchy based on *relationships* and *cross-tree constraints*. A *relationship* is a connection between a parent feature and its child features. A *cross-tree* constraint is a declaration of the type inclusion or exclusion. For instance, if a feature is included/excluded, then another feature must also be included/excluded.

After the introduction of the original FD notation, called Feature-Oriented Domain Analysis method (FODA) [69], an extension, named Feature-RSEB was proposed extending FODA with an additional relationship. Since, the FDs

have been adopted by the SPL community and other extensions suggested such as cardinality-based FDs (introducing cardinalities) and extended FDs (adding attributes).

In this section, first, we describe the basic FDs which are composed on basis of FODA and RSEB notations. Then, we present the cardinality–based FDs and the extended FDs.

2.2.1 Basic feature diagrams

The elements composing a FD are features and relationships between parent features and child features. The root feature represents a concept and the other features define the commonality and variability of this concept [75]. The features in a basic FD can be categorized into four elements:

- Mandatory – A mandatory relationship (illustrated in Figure 2.2) between a parent and a child feature means that the inclusion of the parent feature in the product requires the inclusion of the child feature.
- Optional – An optional relationship (illustrated in Figure 2.3) between a parent and a child feature means that at the inclusion of the parent feature, the child feature may or not be included in the product.
- Or – An or relationship (illustrated in Figure 2.5) between a parent feature and a set of children features means that the inclusion of the parent feature requires the inclusion of one or more child features.
- Alternative – An alternative relationship (illustrated in Figure 2.4) between a parent feature and a set of children features means that the inclusion of the parent feature in the product requires the inclusion of only one child feature.

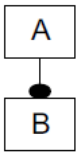


Figure 2.2: Feature diagrams: a mandatory relationship

In addition to the relationships between features, cross-tree constraints allow to restrict feature combinations. The cross–tree constraints in basic FD are of two types:

- Requires – A requires constraint between two features means that the inclusion of a feature implies the inclusion of the feature concerned by the requires constraint. For example, as shown in Figure 2.6, G requires D means that if G is included, then D should be included.
- Excludes – An excludes constraint between two features means that the inclusion of a feature implies the exclusion of the feature concerned by the excludes constraint. The two features cannot be part of the same product. For example in Figure 2.6, E excludes F means that if E is included in a product, then F should not be included and backwards.

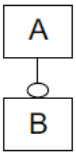


Figure 2.3: Feature diagrams: an optional relationship

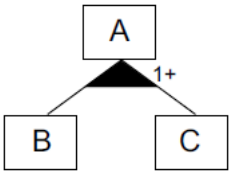


Figure 2.4: Feature diagrams: an or relationship

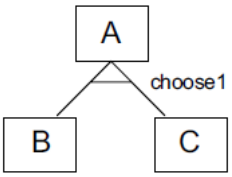


Figure 2.5: Feature diagrams: an alternative relationship

The FD shown in figure 2.7 is an example of basic FD. It presents some features of a car. A *car* possesses a *body*, *transmission*, *engine* and may have

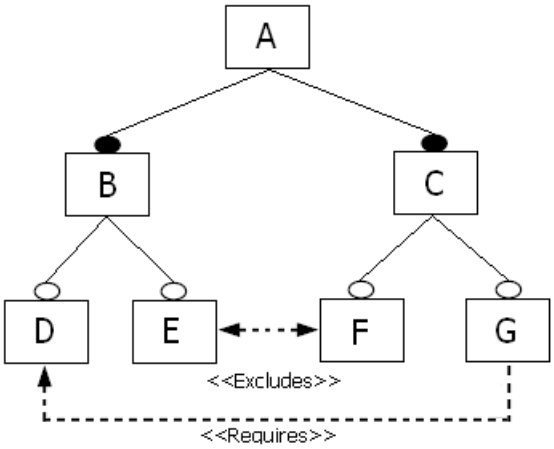


Figure 2.6: Feature diagrams: Requires and Excludes constraints

a *cruise*. The *transmission* is either *automatic* or *manual* but not both. And finally, the engine may be *electric* or *gasoline*, or both.

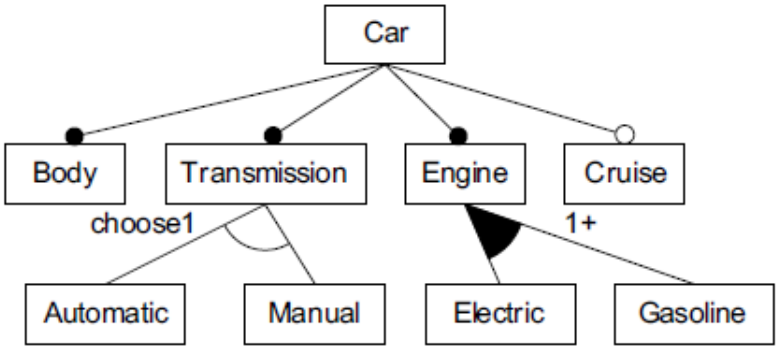


Figure 2.7: Example of feature diagram

2.2.2 Cardinality-based feature diagrams

Some cases, leading to ambiguities in FDs, could not be represented with "alternative" and "or" relationships, so some authors have proposed to extend the basic FDs with cardinalities (also named multiplicities). The cardinalities

in FDs are divided into two types: *group cardinalities* and *feature cardinalities* [63].

The *group cardinalities* (shown in Figure 2.8) can be generalized as a set of features included when their parent feature is included and having a cardinality depending on the number of features of selected features in the set [81]. So, alternative relationships can be interpreted as a [1..1] group cardinality and or relationships can be equivalent to a [1..N] group cardinality in which N is the number of features in the set.

The *feature cardinalities* having the form [N..M] with N the lower bound and M the upper bound, are employed to limit the number of child features each time that their parent feature is selected [63]. When the upper bound is "*", the parent feature has an infinite number of child features as long as the constraints are respected. This notation allows to have products having a random number of components. Mandatory and optional relationships can be represented using cardinalities. Mandatory features have a [1..1] cardinality (as shown in Figure 2.9) whereas optional features have a [0..1] cardinality.

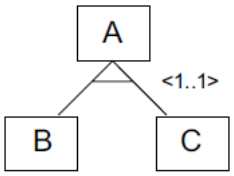


Figure 2.8: Feature diagrams: a group cardinality

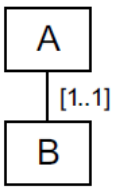


Figure 2.9: Feature diagrams: a feature cardinality

2.2.3 Extended feature diagrams

Basic FDs are useful to represent the commonality and variability of different products in a SPL. But in some cases, additional information about

features have to be added in the model, so the FD must be extended to include attributes. The FDs including additional information (attributes) are named extended FDs [55].

Coping with features attributes introduces four concepts: *feature*, *attribute*, *attribute domain* and *attribute value*. A *feature* is a characteristic of a product. In function of the stage of development, "a feature may refer to a requirement (if products are requirement documents), a component in an architecture (if products are component architectures) or even pieces of code (if products are binary codes in a feature oriented programming approach) of a SPL" [55]. An *attribute* of feature is a property which can be evaluated. The *attribute domain* defines for an attribute the range of its possible values. A domain is always associated to an attribute. An *attribute* value is a value which is part of the domain. When a feature is not selected, it takes a default value. We can distinguish basic attributes and derived attributes, the first are values directly on the domain and the second are attributes of other features combined in expressions. An example of extended FD is presented in Figure 2.10.

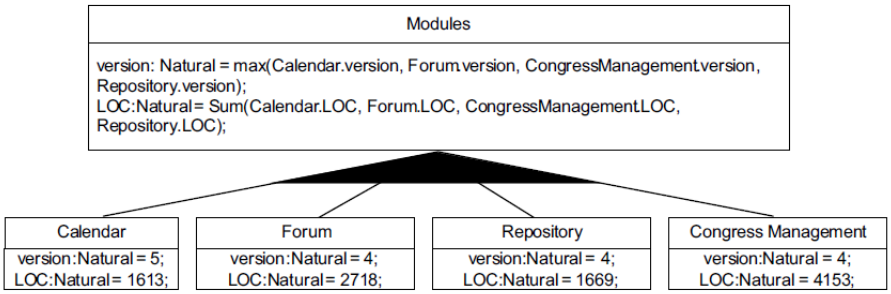


Figure 2.10: Example of extended feature diagram

2.3 Variability modelling and configuration tools for SPL

In this section, we present a selection of variability modelling and configuration tools for SPL. "Variability modelling and configuration tools are tool-support approaches to domain engineering and application engineering activities" [88]. They allow to select and configure the required parts of a product within the SPL. As SPLs are based on commonality and variability, various products can be derived in order to reuse them in software develop-

ment.

First, we present a selection of variability modelling and configuration tools available and then we compare them on different criteria. Finally, we discuss the characteristics of each of them.

2.3.1 Existing variability modelling and configuration tools

We have chosen to present a selection of five current variability modelling and configuration tools: Feature Modelling Plug-in, Kumbang tools, FaMa-FW, RequiLine and Pure::variants. We have selected the Feature Modelling Plug-in, Kumbang tools and RequiLine because they are the most quoted tools in the literature. We have also selected FaMa-FW because it is developed in the Department of Computer Languages and Systems of the University of Seville where we have made our traineeship. As Pure::variants has been recommended to us, we have integrated it into our comparison. The descriptions of these tools are based on the information available in May 2009.

Feature Modelling Plug-in

The Feature Modelling Plug-in (FMP) is a free Eclipse plugin for the edition and the configuration of feature models [15]. FMP is either employed with Eclipse or integrated in the *fmp2rsm* plugin allowing product line modelling in UML. *fmp2rsm* is a plugin for feature-based model templates to IBM Rational Software Modeller [17] or Rational Software Architect [16].

FMP main capabilities are the following: *feature model editor*, *feature-based configurator*, *support for constraints*, *constraint checking and propagation*, *synchronization between feature models and configurations*, *model and configuration exchange*, and *user-extensible metamodel* [62]. The *feature model editor* allows to edit feature models in an explorer-style view (as shown in Figure 2.11) whereas the *feature-based configurator* enables the creation of feature configurations using a check-box view or a wizard. The *support for constraints* uses XPath [50], a query language for selecting nodes or computing values from the content of an *XML* document, and/or propositional formulas to describe additional constraints among features and feature attributes. The *constraint checking and propagation* allows to check the consistency of a feature model which means that it verifies if the feature model has at least one valid configuration. It checks also that concrete configurations satisfy all the constraints from their corresponding feature models. During configurations, the constraint propagation provides an optional guidance (with different levels of assistance) to users in their selections of features that require other features. Those required features will be automatically selected. The system distinguishes the choices that

are made by users, automatically made and undecided. The *synchronization between feature models and configurations* allows to automatically propagate the changes made in a feature model to its configurations. Feature models and configuration are treated in a uniform way. The *model and configuration exchange*, allows to import and export feature models and configurations. The configurations are represented using an *XML* format to be fed into other tools (code generators, etc.). FMP provides also a *user-extensible meta-model* which can be extended with additional information such as priorities, binding times, implementation status, etc. joined to the features. Users define the format of these information completing the feature modelling notations.

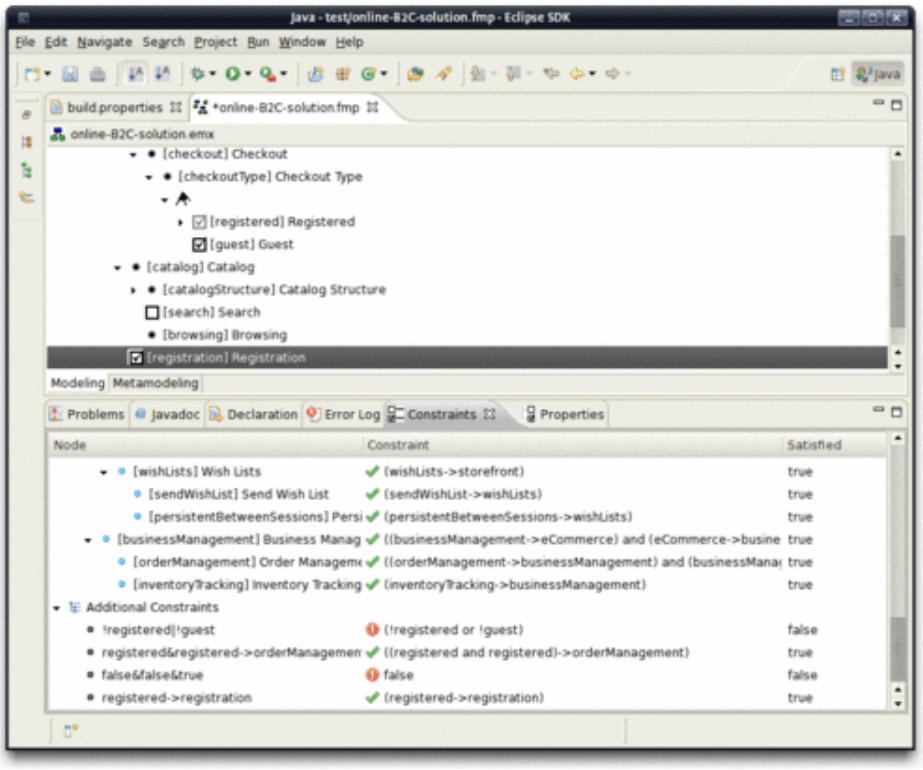


Figure 2.11: Screenshot of the Feature Modelling Plug-in interface

Kumbang tools

Kumbang tools are Eclipse plugins consisting in two tools: *Kumbang Modeller* and *Kumbang Configurator*. Free and open source, they have been con-

ceived for the creation and the configuration of software product families. "Kumbang tools support seamless and integrated domain engineering and application engineering activities" [77].

Kumbang Modeller allows to create and modify models of variability on basis of software product families from features and architectural elements. The graphical interface (visible in Figure 2.12) of the *Kumbang Modeller* leads users through the modelling task in hiding the complexity of concrete syntax. It allows to validate the models made by users in order to check that the derivation of models produces at least one valid product and that there are not other errors such as required interfaces not connected to their corresponding interfaces, constraints not satisfied, and cyclic loops.

Kumbang Configurator allows to bind variability in a model in order to derive individual products configurations. At each new configuration or modification of configuration, their consistency, completeness, and consequences are checked. To be consistent, configurations must have no violation of the rules of the Kumbang model. A complete configuration implies that all mandatory selections are satisfied. The consequences are implications of selections by others and conflicts between selections. The tool must automatically add the implied selections and identify the different conflicts.

Kumbang tools use ontology which has a rigorously-defined semantics [51] to capture the features and components having compositional structure and attributes, the interfaces of components, the connections between the interfaces of components, and constraints to derive product family variability and derived products. A meta-model for Kumbang models and configurations is so offered by the Kumbang ontology. The latter summarizes the modelling of variability in software and non-software product families of the previous approaches.

As Kumbang tools validate automatically the models and configurations, they are particularly useful for software product families having many complex and variability dependencies but also for simpler software product families.

FaMa-FW

"FaMa-FW is a framework for automated analysis of feature models integrating some of the most commonly used logic representations and solvers proposed in the literature (BDD, SAT and CSP solvers are implemented)" [14]. FaMa-FW is presented as an Eclipse plugin and distributed under free license. The main objective pursued by FaMa-FW is to provide an extensible framework to easily integrate and develop current research on variability model automated analysis. Its architecture follows the SPL paradigm, allow-

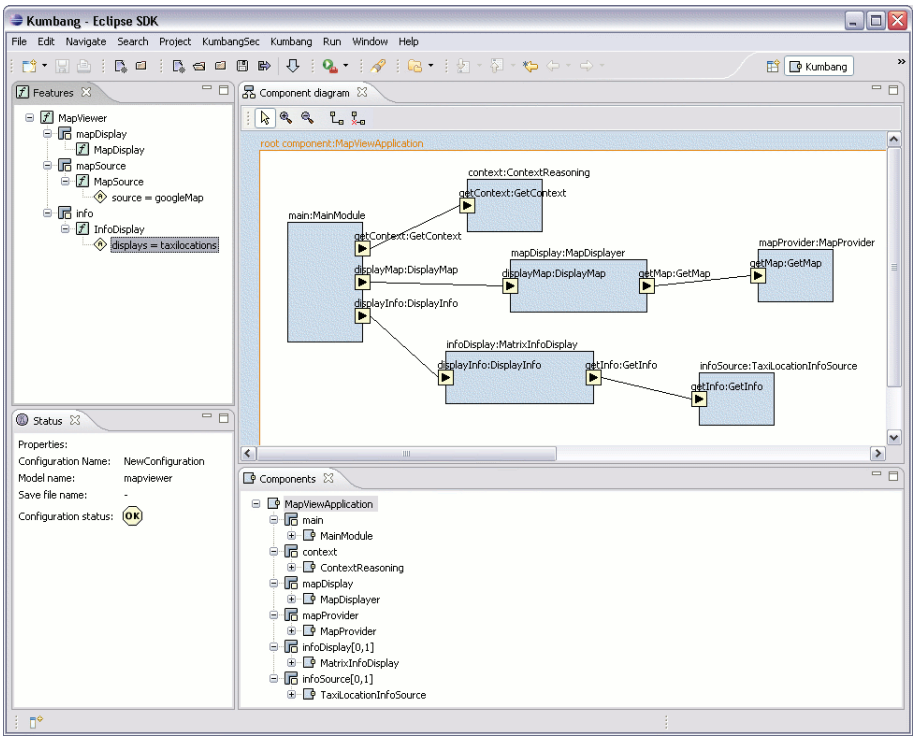


Figure 2.12: Screenshot of the Kumbang tools interface

ing it to support different variability meta-models, reasoners, solvers, analysis questions and reasoner selectors.

The benefits of its architecture are to facilitate its extension by adding new components or features, or updating the existing, the integration of extensions (thanks to its simple and stable interface) and its configuration with a unique *XML* file.

The two main functionalities offered by FaMa-FW are automated model analysis and visual model edition/creation [54]. The model analysis allows once the feature model is created to verify that a feature model is valid (all the constraints satisfied), calculate for a feature model the number of possible products, list possible products of a feature model, calculate the commonality of a feature (number of apparition of a feature in the possible products), etc.

To improve the analysis, different solvers and logic representations can be integrated in FaMa-FW. According to the user needs and the configuration of FaMa-FW, it is able to automatically select the most efficient solver in terms

of performance (execution time). It integrates CSP, SAT and BDD, three solvers for automated analysis of feature models. FaMa-FW can be updated to integrate other solvers.

RequiLine

RequiLine is a requirements engineering tool used in stand-alone for the management of product lines [32]. It uses requirements and feature models for the modelling of product lines and allows from specified models, the derivation of product configurations. RequiLine provides also a graphical editor, a consistency checker, a query interface, a user management (with different views), and an *XML* interface.

RequiLine uses FDs whose the semantics are similar to those of other feature models. It allows to highlight the commonalities and variabilities between the products in the software product family [52].

The main capabilities of RequiLine include the *entering of features and requirements*, a *consistency checker*, and a *query interface* [90]. First, the tool allows to *entering features and requirements*, and defining relationships between them. The interface (visible in Figure 2.13) is divided into three parts, one for the management of products in the product line, one for the management of features and one for the management of requirements. Second, the *consistency checker* analyses that all features and requirement models are consistent and completely specified, which means that they contain the required information (such as state, version, priority, etc.) and dependencies contain no contradictions. A consistent model means that the correctness of domain relationships and dependencies are ensured, the resolving of variation points and the partitioning of features and requirements. The user can select to perform all checks simultaneously or individually by selecting the desired one. The *consistency checker* allows to assist users in their construction of stable and correct feature and requirements models by providing them helpful information. Third, the *query interface* provides a selection of predefined queries for features and requirements models and a query generator. Information about the modelled dependencies and variation points are included into the predefined queries. If these information are not included by the predefined queries, the query generator allows to define new queries.

Pure::variants

Pure::variants is an Eclipse plugin supporting the development and the deployment of SPLs. Pure::variants allows to follow each phase of the entire software configuration process in order to automatically produce valid solutions

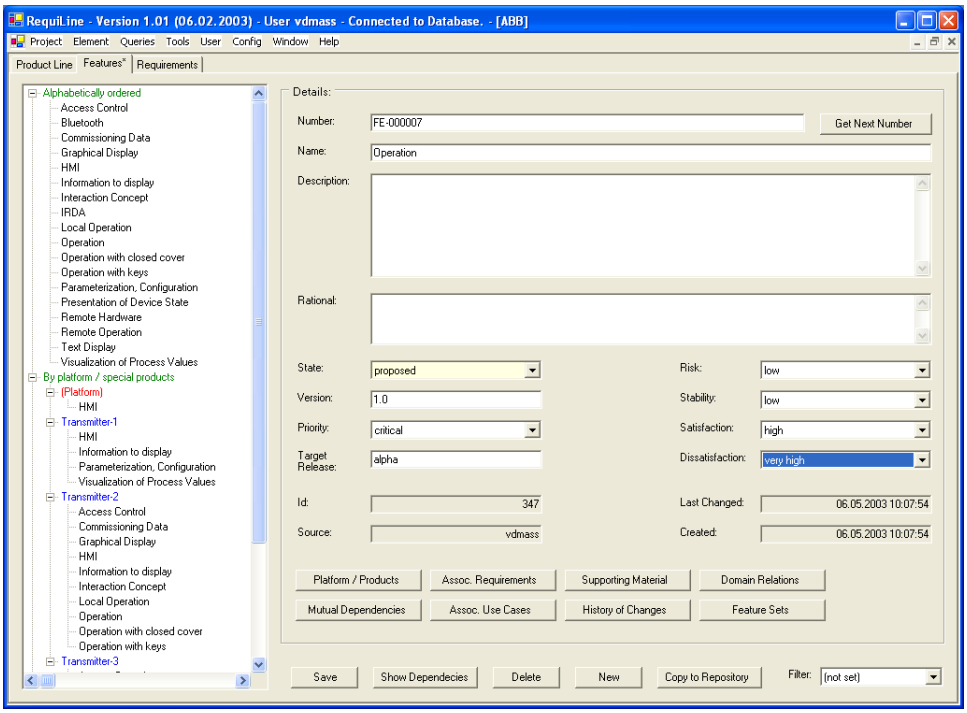


Figure 2.13: Screenshot of the RequiLine interface

from the chosen features, and efficiently manage the components, restrictions and terms of usage of software products [31]. "It has also been designed as an open framework that integrates with other tools and types of data such as requirements management systems, object-oriented modelling tools, configuration management systems, bug tracking systems, code generators, compilers, *UML*, documentation, source code, etc." [79].

Pure::variants allows the development of SPLs using a set of integrated feature models which defines the problem domain, family models and variant description models (VDMs). The representation of the problem domain is made using feature models. The family models are used to represent the solution domain which is the concrete design and implementation of the software family. The VDM specifies individual products from the product line and contains the selected features and their values.

These models represent knowledge which is captured in Pure::variants to provide tool support for all the people involved within a family-based software development process such as the domain analysts, domain designers, applica-

tion analysts, and application developers. Domain analysts use the feature model editor (shown in Figure 2.14), domain designers the family domain editor and application analysts the VDM, whereas application developers use the transformation engine to generate a member of the solution family.

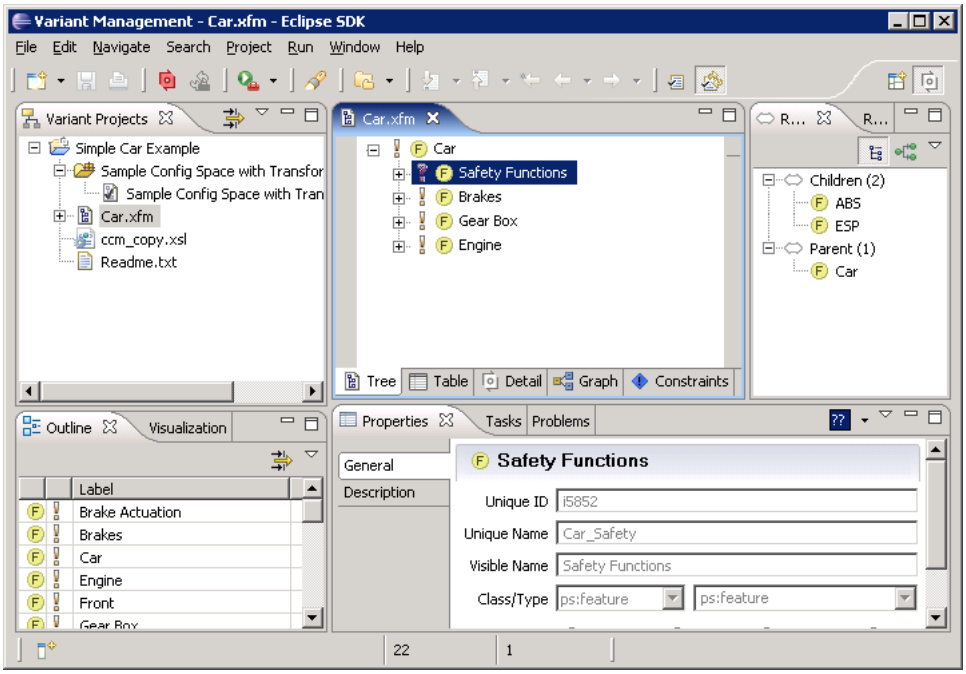


Figure 2.14: Screenshot of the Pure::variants interface

2.3.2 Comparison of selected variability modelling and configuration tools

The comparison is based on a study in which general and technical information for a number of variability modelling and configuration tools for SPL are compared [88]. We confront each tool using the 8 following criteria of comparison:

- **Application type** – Type of application composing the tool.
- **Implementation language** – Implementation language in which the tool has been developed.

- **Supported languages** – Programming languages supported by the tool.
- **Cost** – Required cost to purchase the tool.
- **Product configuration wizard** – Ability of the tool to provide a wizard to help users in the product configuration step.
- **Operating system support** – Capability of the tool to run on the following operating systems: Windows, Mac OS X, Linux, and Unix.
- **Rendering of modelling** – Ability of the tool to render the model in the form of GUI tree, table, box arrow, and textual language.
- **Formats of Input/Output (I/O) models** – Type of format supported by the tool in input and output among *DB*, *HTML*, *XML/XMI*, *CSV*, user define, grammar.

We have chosen these criteria because they are key characteristics allowing to differentiate the different tools. In analysing the application type, we are able to determine the ease of installation of the tools. The implementation and supported languages allow us to highlight the required languages to use the tools and possibly modify them. We are also interested in the cost asked for the use of the different tools because we are looking for a free solution. The functionality providing a product configuration wizard is important to guide users in the process of configuration, increasing the ease of use of the tools. We also examine the operating system supported by the tools because they have to be used on the majority of the operating systems. Another aspect allowing to judge of the ease of use of the tools is the rendering of modelling. Indeed, the representation of the model has an influence on the ease to configure a product and so the ease of use. As last criterion, we are interested by the I/O formats of models supported by the tools because it is important that they support the maximum of formats.

Table 2.1 summarizes the comparison of the different variability modelling and configuration tools on the five first criteria: application type, implementation language, supported languages, cost and product configuration wizard. Four of the five selected tools are Eclipse plugins at the exception of RequiLine which is a stand-alone application. At the level of the implementation language, we can notice that FMP, Kumbang tools and FaMa-FW are written in Java whereas RequiLine and pure::variants has been implemented respectively in *C#* and *.NET*, and in *C/C++*. The supported languages by the tools are dependent of the application type. Indeed, the Eclipse plugins support all

the languages of the Eclipse platform whereas for RequiLine, no information about the supported languages was available. All the selected languages are completely free at the exception of Pure::variants which is free for non commercial use. The last criterion of this table is the ability of the tools to provide a product configuration wizard to help user in their tasks of configuration. Only RequiLine and Pure::variants possess a such functionality.

We continue our comparison by confronting the selected tools to a selection of operating systems, as summarized in Table 2.2. The support of the variability modelling and configuration tools has been tested on Windows, Mac OS X, Linux and Unix. All the tools support to be run on Windows without problems. On Mac OS X, they also run at the exception of Kumbang tools which support partly this operating system. We can make the same observation on Linux. The last operating system is Unix and we observe that Kumbang tools still support partly this platform and that Pure::variants does not support it.

The next comparison criterion is the rendering of modelling and is shown in Table 2.3. The different tools have been tested on four different model representations such as GUI tree, table, box arrow, and textual language. On the one hand, the GUI tree and the textual languages are supported by all the tools. On the other hand, only Pure::variants support tables whereas at the exception of FMP and FaMa-FW, they support box arrow.

The last comparison criterion is the format of I/O models and its results are summarized in Table 2.4. The selection of tools is tested on five formats of (I/O) models: database (*DB*), *HTML*, *XML/XMI*, *CSV*, user define and grammar. If we compare the tools in relation to DB and grammar, we notice that no tools support these formats. For the *HTML*, *CSV* and user define formats, only Pure::variants is able support them. All of the tool support the *XML/XMI* format.

The comparison results are discussed in the next section.

2.3.3 Discussion

We glance through each criterion to identify the main differences between the selected variability modelling and configuration tools in order to identify which one we will use.

On the point of view of the application type, at the exception of RequiLine which is stand-alone, the others are Eclipse plugins. RequiLine has so an advantage because it does not require the install of Eclipse. The implementation languages of FMP, Kumbang tools and FaMa-FW are Java whereas for RequiLine and Pure::variants, it is the C language. It is not possible to differentiate these different tools because they are, according to our experience, the

Table 2.1: Summary of variability modelling and configuration tools comparison: general

Name	Application type	Implementation language	Supported languages	Cost	Product configuration wizard
Feature Modelling Plug-in	Eclipse plugin	Java	All Eclipse platform	Free	No
Kumbang tools	Eclipse plugin	Java	All Eclipse platform	Free	No
FaMa-FW	Eclipse plugin	Java	All Eclipse platform	Free	No
RequiLine	Stand-alone	C#, .NET	No information	Free	Yes
Pure::variants	Eclipse plugin	C/C++	All Eclipse platform	Free for non commercial use	Yes

Table 2.2: Summary of variability modelling and configuration tools comparison: operating systems support

Name	Windows	Mac OS X	Linux	Unix
Feature Modelling Plug-in	Yes	Yes	Yes	Yes
Kumbang tools	Yes	Partly	Partly	Partly
FaMa-FW	Yes	Yes	Yes	Yes
RequiLine	Yes	Yes	Yes	Yes
Pure::variants	Yes	Yes	Yes	No

Table 2.3: Summary of variability modelling and configuration tools comparison: rendering of modelling

Name	GUI tree	Table	Box arrow	Textual language
Feature Modelling Plug-in	Yes	No	No	Yes
Kumbang tools	Yes	No	Yes	Yes
FaMa-FW	Yes	No	No	Yes
RequiLine	Yes	No	Yes	Yes
Pure::variants	Yes	Yes	Yes	Yes

Table 2.4: Summary of variability modelling and configuration tools comparison: formats of Input/Output (I/O) models

Name	DB	HTML	XML /XMI	CSV	User define	Grammar
Feature Modelling Plug-in	No	No	I/O	No	No	No
Kumbang tools	No	No	I/O	No	No	No
FaMa-FW	No	No	I/O	No	No	No
RequiLine	No	No	I/O	No	No	No
Pure::variants	No	I/O	I/O	I/O	I/O	No

most widespread languages which are supposed known by every IT specialist. The supported languages are dependent of the application type. However, for RequiLine, this information was not available so we cannot advantage a tool in relation to the others which are all the languages supported by the Eclipse platform. At the level of the cost, the five tools are equals because they are free of use. An important criterion is the possibility offered by the tools to provide a product configuration wizard. RequiLine and Pure::variants have the advantage that they possess this functionality and not FMP, Kumbang tools, FaMa-FW. On the point of view of the operating system support, as we can see in the Table 2.2, three tools are able to run on the four specified operating systems: FMP, FaMa-FW and RequiLine. On the other hand, there are Kumbang tools which are partly supported by Mac OS X, Linux and Unix, and Pure::variants which is not supported on Unix. The three first have so an advantage on the two others. At the level of the rendering of modelling, we can see that only Pure::variants is able to represent models using the four specified representations (as shown in Table 2.3). The others are not able to represent the model under the form of table or box arrow, or both. Pure::variants has so an advantage on the others. For the last criterion, the formats of (I/O) models, Table 2.4 shows that none of the five tools are able to use all the specified formats. However, Pure::variants support more formats than the four others that support only the *XML/XMI* format.

If we sum up all the advantages of the five tools, we can notice that Pure::variants has more strengths than the others. However, it is disadvantaged on two aspects: application type and operating system support. Its two disadvantages are minor compared to its strengths (I/O format, rendering of modelling, product configuration wizard).

Chapter 3

Running example: A conference website SPL

In this chapter we elaborate on the development of a conference website product line (CWPL) relying on a CMS as support technology. First, we give the basics of conference management and CMSs before defining the characteristics of a CWPL. Then, we apply a requirements engineering template in order to identify the needs for an application generating a CWPL. Finally we explore some of the most important free CMSs available on the market before comparing them and selecting the best CMS for CWPL.

3.1 Basics of conference management

Covering a broad range of specialized sets of topics, conferences allow researchers to present and discuss their work. "Together with academic or scientific journals, conferences provide an important channel to exchange information between researchers" [67]. The conference program is designed to provide maximum opportunity to present high quality papers appropriate to the defined scope of the conference [83]. The work of researchers is presented in the form of concise presentations of 10 to 30 minutes, usually including discussions. It is synthesized under written form, named academic papers, and published as conference proceedings.

During a conference, two types of speakers give talks: keynote speakers and prospective presenters. Keynote speakers, usually prominent scholars, present a lecture during around an hour. Keynotes are advertised before the conference as they usually attest of its importance and quality. Prospective presenters are told to submit their abstracts or papers by a "call for papers" or

a “call for abstracts”. These calls list the conference's topics and are used to attract qualified presenters. Sometimes, presenters base their talk on a visual presentation, like slides, that shows research results and key figures. Some conferences may have only one session at a time, and others, several parallel sessions with speakers in separate rooms speaking at the same time. Abstracts and papers are diligently reviewed and only those matching the quality standard of the conference are accepted for publication. Paper sizes usually vary between 4 and 20 pages and the review process, called peer reviewing, is performed by several experts in the field.

Generally, these peers are members of the *Program Committee* or *Referees*. *Program*, *Publications*, *Publicity* and *Public Relations*, and *Finance Committees* are subcommittees composing the *Conference Committee* responsible for the organization of the conference. The *Program Committee* determines the topic of the conference and organizes the technical program in collaboration with the *Conference Committee* and *Sponsors*. The *Publications Committee* defines the recommendations to publish the complete papers or an abstract of each paper. The *Publicity* and *Public Relations Committee* promote the conference with the objective to get the maximum of attendees. The *Finance Committee* manages the approved budget to ensure that the financial policies are respected.

Typically accommodating more than 250 attendees, conferences are usually organized either by a group of researchers with a common interest or by a scientific society and are held on a regular basis, typically annually. Those characteristics may change because there are different categories of conference: general conferences, themed conferences and professional conferences. A general conference is a conference with sessions covering a wide range of topics. A themed conference is a small conference organized around a particular topic. A professional conference is a large conference not limited to academics but with academically-related issues. In addition, conferences may include, or be held in conjunction with, symposia, workshops or tutorials [83]. A symposium is similar to a conference but covers more specialized topics. With less than 250 attendees, it runs like a conference on one or more days. Symposia may include workshops or tutorials. A workshop is a small meeting limited to a narrow topical area. With no more than 100 attendees, the lodging and meeting space is closely limited to improve communication between them. Often it explores an emerging technology. Workshops may also include tutorials. A tutorial aims at educating a small group of attendees on a selected topic. One or more instructors share their expertise in a specific field.

3.2 Basics of content management systems

A CMS is a tool that enables a variety of (centralised) technical and (de-centralised) non-technical staff to create, edit, manage and publish, in a number of formats, a variety of content (such as text, graphics, video, documents, etc.), whilst being constrained by a centralised set of rules, process and workflows that ensure coherent, validated electronic content [92]. Frequently, CMSs are used for the storage, control, versioning, and publication of industry-specific documentation. Most often, a CMS application will operate from a browser based interface, which will likely have some resemblance to a typical word processing program [93].

Typically, a CMS is made up of two elements: a Content Management Application (CMA) and a Content Delivery Application (CDA). The CMA element allows the content manager or author, who may not know Hypertext Markup Language (*HTML*), to manage the creation, modification, and removal of content from a website without needing the expertise of a webmaster. The CDA element uses and compiles that information to update the website.

The features of a CMS most often include indexing, search, and retrieval, format management, revision control, and web-based publishing [86]. The web-based publishing feature allows to use a set of templates, as well as wizards and other tools to create or modify web content. The format management feature allows legacy electronic documents and scanned paper documents to be formatted into *HTML* or Portable Document Format (*PDF*) for the website. The revision control feature allows content to be updated to a newer version or restored to a previous version. Revision control also tracks any change made to files by individuals. The indexing, search, and retrieval features allow CMSs to index all data within an organization. Individuals can then search for data using keywords, which the CMS retrieves. A CMS may also provide tools for one-to-one marketing [1]. There are hundreds of CMSs available, written in a variety of languages for any platform such as .Net, Java, Flash, Cold Fusion, *PHP*, etc. In Section 3.5, we explore existing CMSs as we have to select one of them for CWPL. Now we have defined the different concepts used in the field of CWPL, a requirements analysis can be made in order to highlight the important characteristics needed to build a CWPL generator.

3.3 Requirements engineering

In this section, we present a first attempt to identify requirements for a new tool allowing to automatically generate a conference website. In this

aim, we have used a Software Requirement Specification (SRS). A SRS is a complete description of the intended purpose and environment for a software to be developed. "It fully describes what the software will do and how it will be expected to perform" [38]. First, we explain how we proceed to complete the specification. Second, we define the structure that we follow in the rest of this section. Third, we describe the stakeholders of the system. Fourth, we define the project constraints to identify how the eventual product must fit into its environment. Finally, we develop the functional and non-functional requirements, which are respectively, the fundamental or essential functions of the product and the behavioural properties that the specified functions must have.

3.3.1 Requirements elicitation approach

To complete a SRS, there are some existing templates available such as IEEE [18], VOLERE [43], Klariti [24], etc. We have chosen the VOLERE Requirements Specification Template because we have completed previously requirements analysis with good results using it. Indeed, we have already used it for the Requirements Engineering course [19] (INFO M431) lavished in the University of Namur by Mr Patrick Heymans. VOLERE will be used to complete the SRS and provide a global view of the system-to-be. We tried to respect the specification structure but the collected information unfortunately turned out to be insufficient to complete entirely the specification. So, we focus then on the scope and the purpose of the system. We omit elements of the specification about which we have no information.

In order to understand better the system and its requirements, we have employed two elicitation techniques to clarify the system: background reading and interviews. During the background reading, we read some documents related to the organization of conferences. This technique allowed us to be ready for other interactive techniques. Then, we made interviews of persons involved in conference organization. They discussed their needs for a system which generates a conference website.

3.3.2 VOLERE template

The VOLERE template is divided into four parts: *Project drivers*, *Project constraints*, *Functional requirements*, and *Non-functional requirements*. We will begin with the project drivers which describe the business-related forces. We will be interested in the *Purpose of the project* and the different people who have a link with the product. Then, we will continue with the *Project constraints*, the constraints on the requirements that will affect the eventual

design of the product. Our interests will be the *Solution design constraints*, the *Partner or collaborative applications* and the *Anticipated workplace environment*. After that, we will tackle the core of this requirement engineering: the *Context of the work* and the *Functional requirements* and *Non-functional requirements*.

3.3.3 Project drivers

The project drivers are the business-related forces. In this section, we focus on the *Purpose of the project* and the different people who have a link with the product. We identify the *client, the customer and other stakeholders* as well as the *Users of the product*.

Purpose of the project

The purpose of the project is to define the user problem and the goals of the project. The *user problem* describes the work context and the situation that triggered the development effort. It also gives a description of the work that the user wants to do with the delivered product. The *goals of the project* are the reasons that the product is being developed.

User problem The active researchers and professors in diversified academic domains need to attend scientific conferences to present and discuss their works and to state their advances. For their participation, they consult the conference website to find the practical modalities such as the submissions deadlines, dates, place, etc. For organizers, the design of the conference website takes a lot of time, that they generally do not have. Organizers who have no knowledge in *HTML*, *PHP*, etc. can call to a professional to set up the conference website, but it costs money which can be devoted to other more important posts in the organization. In the two cases, loss of time and call to a professional, it is problematic. Consequently, a solution has to be developed.

Goals of the project At the present time, there is no tool available on the market allowing to solve the user problem. So the goal of the project is to implement a tool which automatically generates a conference website according to user's needs. The goal is to be able to quickly generate a conference website with the required features in a minimum amount of time.

Client, customer and other stakeholders

Here we define who are the client, customer and other stakeholders of the product and the differences between them. The *client* is the person paying for the development, and owner of the delivered product whereas the *customer* is the person who will buy the product. *Other stakeholders* are the other people or organizations affected by the product.

Clients Mr David Benavides from the Computer Languages and Systems Department (LSI) of the University of Seville (Spain) and Mr Patrick Heymans from the University of Namur (Belgium) are jointly the clients as they will be the owners of the delivered product. They will not pay for the development of the product as it will be developed in the free software philosophy and will consequently be free of use.

Customer The product is built to satisfy the aims of the customers whilst conforming to the constraints of clients. As conference organizers (professors, researchers, engineers or other people) are going to use the product, they can be considered as its customers. Universities can also be customers in getting it for a distribution to all people who organize conferences inside their walls. They will not have to buy the product as it will be free to download and use.

Other stakeholders The people affected by the product could be categorized mainly into four different groups:

- **Conference organizers** – They use the product to generate conference websites. Attendees and conference participants use then the website. They are the user of the product so their involvement and influence are large.
- **Conference contributors** – They use the conference website to submit their papers and to consult information as the submission deadline, acceptance notification, etc. They have no contact with the product, they just consult and use the functions of its result (i.e. the conference website).
- **Conference participants** – They just consult web pages of the conference website to get practical information or to register to the conference. They have no contact with the product, they just consult its result (i.e. the conference website).

- **System administrators** – They are responsible for all modifications or evolutions of the product so their involvement and influence are large.

Users of the product

We define the *Hands-on users of the product* as well as the *Priorities assigned to them* and their *Participation*. Then we introduce the *Maintenance users*. As we have seen, the role of the client is to pay for the development of the product and the role of the customer is to buy the product. The role of *hands-on users* is to use the product to do work. We will use the characteristics of those users to define the usability requirements for the product. The *Priorities assigned to users* are important because some users must be considered to be more important to the product, or the organization. This should then be clearly defined because it should affect the way that the product is designed. Some users must be listed as having no impact on the product. It means that they will make use of the product, but have no vested interest in it. Any special requirement from these users will have a lower design priority. So we can attach to each category of users a priority rating. Many projects fail through lack of *User participation*, sometimes this is because the required degree of participation was not made clear. It is important that specified user resources are allocated to the project. So we attach to each category of users, a statement of the participation that will be necessary for them to provide the requirements. *Maintenance users* are a special type of hands-on users who have requirements that are specific to maintaining and changing the product. They will help to trigger requirements that might otherwise be missed.

The hands-on users of the product We can identify two categories of users who use the system: the conference organizers and the system administrators. Attendees and conference participants deal with the results provided by the product but not with it, so they cannot be considered as users of the system. For the two categories, we provide the following information:

- **User role** – Summary of the users' responsibilities.
- **Subject matter experience** – Summary of the users' knowledge of the business.
- **Technological experience** – Description of the users' experience with relevant technology.
- **Other user characteristics** – Description of any characteristic of the user that have an effect on the requirements and eventual design of the

product. This description is made on characteristics such as: physical abilities/disabilities, intellectual abilities/disabilities, education, linguistic skills, age group, and gender.

As shown in Table 3.1, the conference organizers are the main users of the system as they provide the main input. Their importance lies within the fact that they are responsible of completing the fields with the information corresponding to the conference website. System administrators are in charge of the system maintenance and information checking. Both must have a master subject matter and technological experience, and no physical disabilities that prevent from using the system. They also have high standard intellectual abilities. As conferences take place in the academic field or between professionals, conference organizers have often a university level or a high standard. System administrators have a high standard education because they have made a minimum of studies in the computer science field to administrate the system. The Graphical User Interface (GUI) and the documentation will be in English. Consequently, a minimum knowledge of this language is required for the two categories. As conference organizers have a high standard or university level, we can evaluate the minimum age at 25 years old and the maximum to 65 years old. System administrators have a lower education level, so the age goes from 20 to 65 years old. Finally, for both, there is no distinction in the gender, they are male or female.

Priorities assigned to users We attach to each category of user described previously a priority rating. We prioritize the users into 3 categories: key users, secondary users and unimportant users. *Key users* are critical to the continued success of the product. We will give a greater importance to requirements generated by this category of users. *Secondary users* use the product but their opinion of it has no effect on its long-term success. *Unimportant users* represent the category for which we give the lowest priority. It includes infrequent, unauthorized and unskilled users, and people who misuse the product. For each category of users, we give a percentage to assess the amount of consideration given to this category.

Key users They are of two types:

- Conference organizers
- System administrators

These key users represent 100% of the totality of users using the product.

Table 3.1: Requirement engineering: summary of the hand-on users of the product

Information		**Conference organizers**	**System administrators**
Role		They are the main users of the system. They are responsible of completing the fields with the information corresponding to the conference website.	They are in charge of the system maintenance and information checking.
Subject matter experience		Master	
Technological experience		Master	
Other user characteristics	Physical abilities/disabilities	No physical disabilities that prevent from using the system.	No physical disabilities that prevent from using the system.
	Intellectual abilities/disabilities	High standard.	High standard.
	Education	University level or high standard.	High standard.
	Linguistic skills	The graphical interface will be in English so a minimum knowledge of this language is required.	English
	Age group	25 to 65 years old.	20 to 65 years old.
	Gender	Male or female.	Male or female.

Secondary users
No secondary users for this product.

Unimportant users
Conference contributors and participants as they simply consult the website but do not use the product.

User participation The participation of key users (see above) is fundamental to provide the different requirements. We can take advantage of their business knowledge of conferences. They can give advices and their point of view on the requirements of the product, just as for its design, mainly for the user interface.

Maintenance users The system administrators working for the client (David Benavides and Patrick Heymans) are charged to maintain and change the product.

3.3.4 Project constraints

The project constraints are constraints on the requirements that will affect the eventual design of the product. They are of 3 types: *solution design constraints*, *partner or collaborative applications* and *anticipated workplace environment*. *Solution design constraints* specify constraints on the way that the problem must be solved. The client, customer or users may have design preferences. These preferences are constraints that must be part of the final product and if these are not met then the solution is not acceptable. *Partner or collaborative applications* are applications that are not part of the product but with which the product will collaborate. We provide information about design constraints that are caused by using partner applications. *Anticipated workplace environment* is the workplace in which the users will work and use the product. This should describe any feature of the workplace that could have an effect on the design of the product.

Solution design constraints

We have identified two design constraints that must be part of the final product. For each of them, we provide the following information:

- **Description** – Sentence statement of the intention of the constraint.
- **Rationale** – Justification of the constraint.

- **Fit criterion** – Measurement of the constraint such that it is possible to test if the solution matches the original constraint.

Table 3.2 describes the first constraint which imposes that the product will be multi-platform because users have to run the tool on their computer or an external computer with different operating systems such as Windows, Mac OS X, Linux or Unix. To test if the solution matches the original constraint, the system must run on the three operating systems mentioned above.

The second constraint described in Table 3.3 is that the product must use a CMS. For a better management, the website generated by the product must be of CMS type. The fit criterion is that the CMS generated by the product must be ready to use.

Table 3.2: Requirement engineering: summary of the first constraint

Constraint #: 1 **Description:** The product must be multi-platform. **Rationale:** Users must be able to run the tool with their computer or an external computer with Windows, Mac OS X, Linux or Unix operating systems. **Fit criterion:** The system must run on the four operating systems mentioned above.

Table 3.3: Requirement engineering: summary of the second constraint

Constraint #: 2 **Description:** The product must use a CMS. **Rationale:** For a better management, the website generated by the product must be of CMS type. **Fit criterion:** The CMS generated by the product must be ready to use.

Partner or collaborative applications

In some cases, the product would collaborate with conference management systems such as EasyChair [13], OpenConf [28], Colibri [11], etc. that supports the organization of conferences. They help the program committee, conference organizers, contributors and reviewers (see Section 3.1 for more details about the participants) in their respective activities. They allow the submission management, paper assignment, paper review, committees management, etc.

Anticipated workplace environment

The workplace can be anywhere an Internet connection is available. To use the system, an Internet connection is not required but once the generation is finished, the files upload requires a connection.

3.3.5 Functional requirements

First, the *context of the work* has to be presented in order to have a better understanding of the *functional requirements*. The *context of the work* identifies the work that is needed to investigate in order to be able to build the product. "*Functional requirements* specify the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business requirements" [91]. The boundaries for the work study and requirements effort must be clearly defined. Without this definition, there is little chance of building a product that will fit seamlessly into its environment.

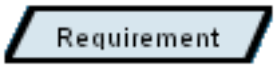
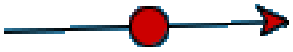
Context of the work

Elicitation techniques (interviews and reading) allow to have a better understanding on the work context. We define it through a work context diagram (*goal model*) using the KAOS language. KAOS is an approach for goal-oriented requirements engineering having many formal analysis techniques [72]. Analysts are able to build requirements model and to derive requirements documents from KAOS models [59].

In our goal model, five components are used: *goal*, *requirement*, *agent*, *goal refinement*, and *responsibility*. A *goal* is an objective that must be met by the system to develop. It requires the cooperation of agent to be reached. A *requirement* is a goal of low level which can be placed under the responsibility of an agent which is part of the system to develop. An *agent* is either human beings, automated components or a software that are responsible for achieving requirements. The *goal refinement* is a relation allowing to point out how a goal (more general) can be satisfied by a set of sub-goals (more specific) and properties of the domain. The *responsibility* is a relation used to assign a requirement to an agent [60]. The legend allowing to understand our KAOS diagram is shown in Table 3.4.

Our goal model is composed of four different models which identify the goals and requirements of the conference organizers, conference contributors and participants. The identified requirements related to the conference website must be part of the final system.

Table 3.4: Requirement engineering: legend of the goal model

Legend	
Type	**Object**
Goal	
Requirement	
Agent	
Goal refinement	
Responsibility	

The first model (shown in Figure 3.1) shows the goals of the conference organizers. To complete the *Organize a conference* goal, other sub-goals must be reached such as: *Attract the appropriate papers* which are in the scope of the conference, *Attract qualified contributors* and *Generate the conference website*. This last goal can be refined in another model which is presented in Figure 3.2. Its objective, completed thanks to all the requirements and goals that we will describe below, is to generate the conference website (*Generate the conference website* goal). Each conference has a title, a place and dates. These are mandatory information that must be imposed. We can add three requirements to our goal model: *Impose to select title*, *Impose to select place*, *Impose to select dates*. In addition to the conference dates, other events can take place such as different deadlines (registration, submission, etc.), discussants assignment, etc. This can be satisfied by the addition of the requirement *Propose to add different events*. Once all the dates are defined, users asked to have the possibility to save them using a standardized form to import it in a calendar (requirement *Propose to save dates in a standardized form*). They may also be interested presenting information of the conference in different languages. As English is the most commonly used language during international conferences, it will be probably the most selected language. Some conferences take place in their languages, it is so interesting to propose different other languages such as Spanish, French, Dutch and German. In some countries as Belgium, there are different national languages (French, Dutch and possibly German), therefore propose up to 3 languages for the conferences seems to be sufficient. We can thus add the two following requirements: *Propose 5 different languages* and *Impose the selection of minimum 1 and maximum 3 languages*. Often,

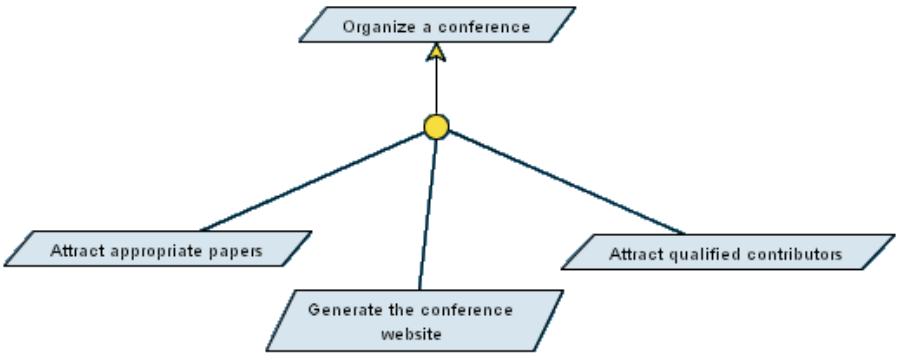


Figure 3.1: Goal model 1: organize a conference

conferences or organizing institutions have a logo which can be displayed to illustrate the conference. It is important to note that this requirement proposes the display but does not impose it. We can thus add the requirement *Propose to display the conference logo*. With a unique template, it is impossible to satisfy all the conference organizers on the point of view of the appearance of the website. The addition of different templates to personalize the website is opportune (requirement *Propose to add additional templates*). Some users asked to display images, for example to illustrate the place of the conference. It is interesting to add more than 2-3 images so we have decided to display until 10 images. For that reason, we can add to our goal model the two following requirements: *Propose to display slideshow* and *Display until 10 images in the slideshow*. To inform persons interested by the conference, we have two means: newsletters sent by email to registered persons and news displayed on the conference website. A newsletter and a news systems are thus required (requirements *Propose a newsletter system* and *Propose a news system*). Often persons wishing to attend the conference must enrol. For that, different personalized forms can be added. We can thus add the requirement *Propose a forms system* to the goal model. When a date has to be added, there is always a problem of format: some people give the date in numbers whereas others in letters. In order to avoid this problem and in an aim of standardization, it is easier to choose a date in a calendar. We can add the requirement *Propose a calendar to select dates*. All the requirements we have expressed participate to fill the *Build a product of SPL* goal. Requirements beginning with "Impose" might be completed by users and those beginning with "Propose" will be at their choice. Once the product is completed, its validation is required

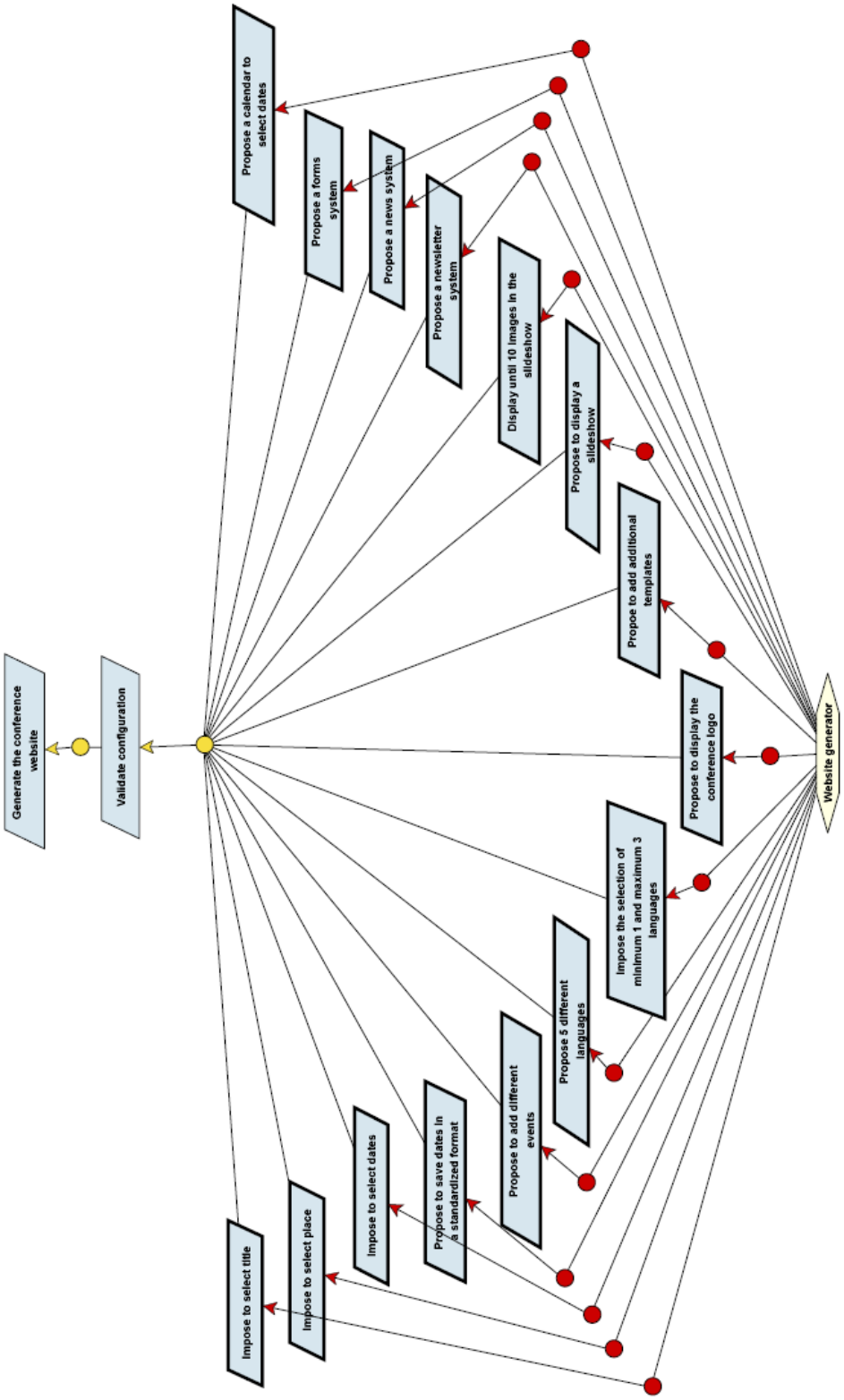


Figure 3.2: Goal model 2: generate the conference website

to check that it is correct (requirement *Validate configuration*). The previous goals participate to fill this requirement.

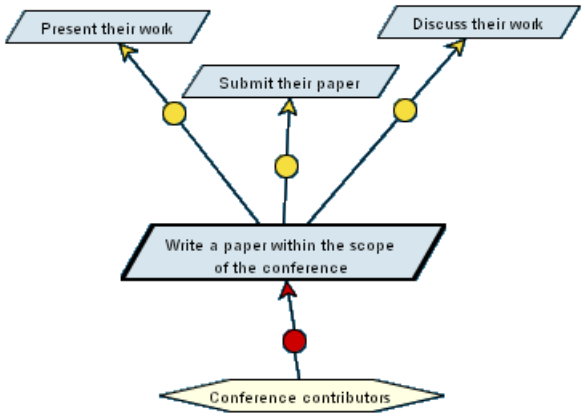


Figure 3.3: Goal model 3: goals of the contributors

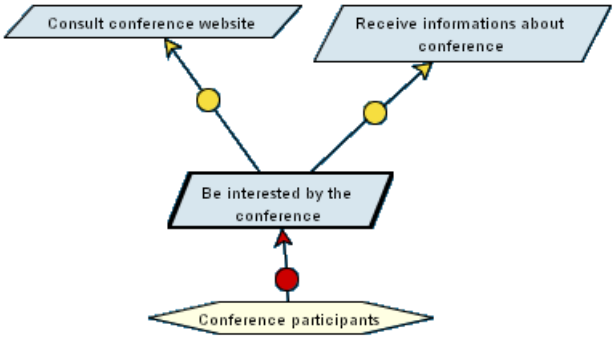


Figure 3.4: Goal model 4: goals of the participants

The third model (shown in Figure 3.3) represents the goals of the conference contributors. To reach their goals, they have to write a paper which is within the scope of the conference (*Write a paper within the scope of the conference* requirement). They can before the conference, *Submit their paper* and during it, *Present their work* and *Discuss their work*.

The fourth model (visible in Figure 3.4) describes the goals of the con-

ference participants. If they are interested by the conference (*Be interested by the conference* requirement), they can *Consult the conference website* and *Receive information about the conference.*

Functional requirements

From Table 3.5 to Table 3.16, we specify the detailed functional requirements introduced in the previous section that must be supported by the product. For each requirement, we provide the following information:

- **Goal model** – Goal model that needs the requirement.
- **Description** – Sentence statement of the intention of the requirement.
- **Rationale** – Justification of the requirement .
- **Source** – Technique which allows to raise the requirement.
- **Fit criterion** – Measurement of the requirement such that it is possible to test if the solution matches the original requirement.
- **Customer satisfaction** – Degree of stakeholder happiness if the requirement is successfully implemented.
- **Customer dissatisfaction** – Degree of stakeholder unhappiness if the requirement is not part of the final product.

The first requirement (Table 3.5), highlighted by background reading, imposes that the product validates each configuration of conference website made by users. Indeed, when users select features which should be available on their conference website, the product must check the validity of the selection in relation to the CWPL. To test if the solution we will provide matches this requirement, the product shall generate a percentage of 100% of valid configurations (during a month of use). As the validation is necessary for the generation, we give 5 as customer satisfaction degree. Without validation, the conference website will never be correct so the customer dissatisfaction can be evaluated to 5 because the validation is mandatory.

The second requirement (shown in Table 3.6) has been mentioned by users during interviews and imposes that the product offers to users the possibility to display the conference logo on the website. Conferences often have their own logo which can be displayed if it is available. To test if the solution we will provide matches this requirement, in one month's use, the product shall copy all the logo files without any error. This requirement is really useful, so

Table 3.5: Requirement engineering: requirement 1

Requirement #: 1	**Goal model #:** 2
Description: The product shall validate each configuration of conference website made by users.	
Rationale: When users select features which should be available on their conference website, the product must check validity of the selection in relation to the conference website product line.	
Source: Background reading.	
Fit criterion: In one month's use, the product shall generate a percentage of 100% of valid configurations.	
Customer satisfaction: 5	
Customer dissatisfaction: 5	

if it is implemented in the final product, the customer satisfaction will be of 4. On the other hand, it will be of 3 if it is not part of the final product.

Table 3.6: Requirement engineering: requirement 2

Requirement #: 2	**Goal model #:** 2
Description: The product shall offer to users the possibility to display the conference logo on the templates.	
Rationale: Conferences often have their own logo which can be displayed if it is available.	
Source: Interviews of users.	
Fit criterion: In one month's use, the product shall copy all the logo files without any error.	
Customer satisfaction: 4	
Customer dissatisfaction: 3	

As shown in Table 3.7, the third requirement imposes that the product proposes 3 languages among 5 languages as website languages, one by default and 2 additional. As the product can be used by everybody in the world, it must propose different languages as default language for the website: English, Spanish, French, Dutch and German. Two other languages among them can be added to the default language. This requirement has been highlighted by interviews of users. To test if the solution we will provide matches it, in one month's use, the product shall generate the website in the selected language(s) without any error. The customer satisfaction and dissatisfaction can be evaluated respectively to 3 and 2. As English will probably be the most selected language, the choice of other languages is a plus. Users will be not

much affected if the requirement is not implemented.

Table 3.7: Requirement engineering: requirement 3

Requirement #: 3	**Goal model #:** 2
Description: The product shall propose 3 languages among 5 languages as website languages, one by default and 2 additional.	
Rationale: As the product can be used by everybody in the world, it must propose different languages as default language for the website: English, Spanish, French, Dutch and German. Two other languages among them can be added to the default language.	
Source: Interviews of users.	
Fit criterion: In one month's use, the product shall generate the website in the selected languages without any error.	
Customer satisfaction: 4	
Customer dissatisfaction: 2	

The fourth requirement (shown in Table 3.8) asked by users imposes that the product offers the possibility to add a minimum of 6 events in addition to the conference dates. Indeed, during a conference organization, events such as registration or submissions deadlines can take place. So it is useful to add these activities and their corresponding dates. To test if the future solution matches this requirement, in one month's use, the product shall add the selected events in the dates website page without any error. If the product does not allow to add these events, users have to manually add them after the generation. For this reason, a customer satisfaction of 4 can be given and a customer dissatisfaction of 3 because the manual addition is not a difficult operation.

Table 3.8: Requirement engineering: requirement 4

Requirement #: 4	**Goal model #:** 2
Description: The product shall offer the possibility to add a minimum of 6 events in addition to the conference dates.	
Rationale: During a conference organization, events such as registration or submissions deadlines can take place. So it is useful to add these activities and their corresponding dates.	
Source: Interviews of users.	
Fit criterion: In one month's use, the product shall add the selected events in the dates website page without any error.	
Customer satisfaction: 4	
Customer dissatisfaction: 2	

As shown in Table 3.9, the fifth requirement, highlighted by interviews, imposes that the product proposes a calendar to easily choose dates or events of the conference. The use of a calendar is more user-friendly for users because they can select the date on the calendar rather than add it manually. To test if the solution we will provide matches this requirement, in one month's use, the product shall recover the selected date in the calendar without any error. The addition of a date will be facilitated if the requirement is implemented in the final product. We can give an evaluation of 4 as customer satisfaction. In addition, this requirement allows to have a standardization of all dates. The customer dissatisfaction will be high (evaluated to 4) if it is not implemented because if the dates formats are different, it will cause verification problems.

Table 3.9: Requirement engineering: requirement 5

Requirement #: 5 **Goal model #:** 2
Description: The product shall propose a calendar to easily choose dates or events of the conference.
Rationale: The use of a calendar is more user-friendly for users because they can select the date on the calendar rather than add it manually.
Source: Interviews of users.
Fit criterion: In one month's use, the product shall recover the selected date in the calendar without any error.
Customer satisfaction: 4
Customer dissatisfaction: 4

As shown in Table 3.10, the sixth requirement imposes that the product offers the possibility to save all dates in a standardized format. The use of a standardized format allows to share, send by email and import dates in a calendar. This requirement has been asked by users (interviews). In one month's use, all the dates inserted by the product in the standardized format must be recovered without any error. We can evaluate the customer satisfaction and dissatisfaction respectively to 3 and 2 because the requirement is useful but probably not often used.

Table 3.11 describes the seventh requirement, highlighted by background reading and interviews of users, imposes that the product shall offer the possibility of adding a plugin managing a newsletter. Some users can be interested in receiving different information by email. That notification can, for example, be sent to the subscribers of the newsletter. To test if the provided solution matches this requirement, all the registered persons must receive the newsletters during one month's use. The degree of customer satisfaction can be evaluated to 4 because a newsletter is a good means to inform interested

Table 3.10: Requirement engineering: requirement 6

Requirement #: 6	**Goal model #:** 2
Description: The product shall offer the possibility to save all dates in a standardized format.	
Rationale: The use of a standardized format allows to share, send by email and import dates in a calendar.	
Source: Interviews of users.	
Fit criterion: In one month's use, all the dates inserted by the product in the standardized format must be recovered without any error.	
Customer satisfaction: 3	
Customer dissatisfaction: 2	

persons. Other means might be less practical and more expensive but possible, so the customer dissatisfaction can be evaluated to 3.

Table 3.11: Requirement engineering: requirement 7

Requirement #: 7	**Goal model #:** 2
Description: The product shall offer the possibility of adding a plugin managing a newsletter.	
Rationale: Some users can be interested in receiving different information by email. That notification can, for example, be sent to the subscribers of the newsletter.	
Source: Background reading and interviews of users.	
Fit criterion: In one month's use, all the registered persons must receive the newsletters.	
Customer satisfaction: 4	
Customer dissatisfaction: 3	

As shown in the Table 3.12, the eighth requirement imposes that the product offers the possibility of adding a plugin managing news related to the conference. Users must be informed of all novelties of the conference by displaying the news on the website. Background reading and interviews of users allowed us to highlight the needs for this requirement. To test if the future solution matches it, in one month's use, the users must have access to the news displayed on the website. We can evaluate the customer satisfaction to 4 and the customer dissatisfaction to 3 for the same reasons as the previous requirement.

The ninth requirement (Table 3.13) has been highlighted by background reading and interviews of users and imposes that the product shall offer the

Table 3.12: Requirement engineering: requirement 8

Requirement #: 8	**Goal model #:** 2
Description: The product shall offer the possibility of adding a plugin managing news related to the conference.	
Rationale: Users must be informed of all novelties of the conference by displaying the news on the website.	
Source: Background reading and interviews of users.	
Fit criterion: In one month's use, the users must have access to the news displayed on the website.	
Customer satisfaction: 4	
Customer dissatisfaction: 3	

possibility of adding a plugin managing forms. On a website, users often have to fill in a form or a questionnaire. For this task, it can be useful to integrate mail forms to the website. To test if the future solution matches this requirement, it must be possible to add forms to the website during one month of use. A conference website often need to register persons for different reasons (participation, paper submission, etc.), so this requirement is very useful (customer satisfaction of 4). On the other hand, if this requirement is not part of the final product, the customer dissatisfaction can be evaluated to 4 because there is no other means for automatic registration.

Table 3.13: Requirement engineering: requirement 9

Requirement #: 9	**Goal model #:** 2
Description: The product shall offer the possibility of adding a plugin managing forms.	
Rationale: On a website, users often have to fill in a form or a questionnaire. For this task, it can be useful to integrate mail forms to the website.	
Source: Background reading and interviews of users.	
Fit criterion: In one month's use, it must be possible to add forms to the website.	
Customer satisfaction: 4	
Customer dissatisfaction: 4	

As shown in Table 3.14, the tenth requirement imposes that the product offers the possibility to add a plugin displaying a slideshow on the homepage. Webmasters can be interested in displaying a serie of chosen images on the homepage of their website. We have identified this requirement thanks to

background reading and interviews of users. The solution that we will provide can be tested if in one month's use, the slideshow must be visible on the website and display the selected images. This requirement is not essential, a majority of users will probably not use it, so the customer dissatisfaction can be evaluated to 2. Nevertheless, it will be useful (for example, to present the place of the conference), so we attribute 3 as customer satisfaction.

Table 3.14: Requirement engineering: requirement 10

Requirement #: 10 **Goal model #:** 2
Description: The product shall offer the possibility to add a plugin displaying a slideshow on the homepage.
Rationale: Webmasters can be interested in displaying a serie of chosen images on the homepage of their website.
Source: Background reading and interviews of users.
Fit criterion: In one month's use, the slideshow must be visible on the website and display the selected images.
Customer satisfaction: 3
Customer dissatisfaction: 2

Table 3.15 shows the eleventh requirement has been highlighted by users (interview). It imposes that the product offers the possibility to add up to 10 images in the slideshow. Generally, a slideshow must display a set of images and is not limited to some images. To test if the solution we will provide matches this requirement, in one month's use, the product shall copy all the selected images without any error. The customer satisfaction can be evaluated to 3 because it is more practical to add images via the product than manually later. In addition, it is more interesting to add until 10 image rather than 2 or 3. The customer dissatisfaction can be evaluated to 2 because it is always possible to add images afterwards.

As shown in Table 3.16, the twelfth requirement imposes that the product offers the possibility of adding 5 other templates in addition to the default template. Indeed, users can be interested in changing the look and feel of their website in order to personalize it. This requirement has been asked by users during interviews. To test if the solution matches it, we test if in one month's use, the product adds the selected templates without any error. We can evaluate the degree of customer satisfaction and dissatisfaction to, 5 and 3 respectively, because it is important and useful to personalize the look and feel of the website. We have attributed an average degree of dissatisfaction because this operation can be made once the generation is completed and the website put in place.

Table 3.15: Requirement engineering: requirement 11

Requirement #: 11	**Goal model #:** 2
Description: The product shall offer the possibility to add up to 10 images to the slideshow.	
Rationale: Generally, a slideshow must display a set of images and is not limited to some images.	
Source: Interviews of users.	
Fit criterion: In one month's use, the product shall copy all the selected images without any error.	
Customer satisfaction: 3	
Customer dissatisfaction: 2	

Table 3.16: Requirement engineering: requirement 12

Requirement #: 12	**Goal model #:** 2
Description: The product shall offer the possibility of adding 5 other templates in addition to the default template.	
Rationale: Users can be interested in changing the look and feel of their website in order to personalize it.	
Source: Interviews of users.	
Fit criterion: In one month's use, the product shall add the selected templates without any error.	
Customer satisfaction: 5	
Customer dissatisfaction: 3	

3.3.6 Non-functional requirements

In addition to the functional requirements, the non-functional requirements include descriptions of quality attributes and performance goals. The description of the product's characteristics in various dimensions that are important to users or developers allows to describe the product's functionalities. The description of the product's functionalities is increased by the quality attributes [91]. Here we focus on the *look and feel*, *usability and humanity*, *performance*, and *operational* requirements.

Look and feel requirements

The look and feel requirements deal with the interface and the style of the product. The requirements on the *interface* ensure that the appearance of the product conforms to the organization's expectations. We capture the

requirements for the interface rather than the design of the interface. The requirements on the *style of the product* give a description of salient features of the product that are related to the way a potential customer will see the product. Once the functional requirements are satisfied, it is often the appearance of products that determines whether they are successful or not. We determine how the product shall appear to its intended consumers.

The interface The product shall have an attractive and practical interface for a scientific use. The succession of screens shall be practical and coherent. The interface shall be clear with no ambiguity on the navigation and provide feedback.

The style of the product The product must have a professional appearance and comply with the standard of the organizing institution.

Usability and humanity requirements

Usability and humanity requirements define the ease of use and the ease of learning of the product. They also describe understandability and politeness, and accessibility requirements. *The ease of use* section describes the client's aspirations for how easy it will be for the intended users of the product to operate it. The product's usability is derived from the abilities of the expected users of the product and the complexity of its functionalities. *Ease of learning* gives a statement of how easy it should be to learn to use the product. We quantify the amount of time that the client feels is allowable before a user can successfully use the product. This requirement will guide designers in how users will learn the product. *Understandability and politeness requirements* specify the requirements for the product to be understood by its users. While usability refers to ease of use, efficiency etc., understanding determines whether the users instinctively know what the product will do for them. *Accessibility requirements* define how easy the product should be for people with common disabilities to access the product. These disabilities might be to do with sight, physical disablement, hearing, cognitive, or others.

Ease of use As described in Table 3.17, we have identified different requirements of usability. For each of them, we define a fit criterion, a measurement of the requirement such that it is possible to test if the solution matches the original requirement.

As the product is made for generating conference websites in a specific domain, the first usability requirement is that the product is efficient to use

Table 3.17: Requirement engineering: summary of the usability requirements

#	**Description**	**Fit criterion**
1	The product is made for generating conference websites in a specific domain, so the product shall be efficient to use for all users of that domain. The user shall quickly be able to use the product.	The user shall be able to understand all functions and use the system after 10 minutes.
2	The interface shall be clear and coherent so the ease of remembering shall be maximum: the user shall quickly remember how to use the product.	The user shall be able to understand the interface after 10 minutes.
3	The error rate shall be minimum because the product shall provide feedback for all errors that the user will make.	The user should not execute understanding errors after a week of use. However, (s)he will be susceptible to make inattention errors.

for all users of that domain. The user shall quickly be able to use the product. To test if the solution meets this requirement, all functions and the use of the system must be understand by the user after 10 minutes. The second requirement is that the interface must be clear and coherent, so the ease of remembering must be maximum: the user must quickly remember how to use the product. Test if the user is able to understand the interface after 10 minutes is the best means to note that the solution matches this requirement. The last usability requirement concerns the error rate. It must be minimum because the product must provide feedback for all errors that the user make. We test this solution during a week of use in which the user should not execute understanding errors. However, (s)he is susceptible to make inattention errors.

Ease of learning The product shall be useable by any user without any training. It shall allow users to generate their first conference website in a few minutes.

Understandability and politeness requirements To be the most intuitive and easy to use, the product shall hide the details of its construction from the user. The product shall also use symbols and words that are naturally

understandable by the users of the system.

Accessibility requirements The product shall be usable by users with physical disabilities or partially-sighted users if they are equipped with necessary tools: special keyboards or keypads, pointing device, etc.

Performance requirements

Performance requirements define the speed and latency, reliability and availability, and robustness or fault tolerance requirements. *Speed and latency requirements* specify the amount of time available to complete specified tasks. *Reliability and availability requirements* quantify the necessary reliability of the product. This is usually expressed as the allowable time between failures, or the total allowable failure rate. It also quantifies the expected availability of the product. *Robustness or fault tolerance requirements* specify the ability of the product to continue to function under abnormal circumstances. They allow to ensure that the product is able to provide some or all of its services after or during some abnormal happening in its environment.

Speed and latency requirements The time to go from a screen to another shall be of maximum 1 second. When the user is familiar with the system, (s)he shall be able to generate a website in maximum 3 minutes.

Reliability and availability requirements Once it has been installed on the computer of the user, the product shall be available 24 hours a day. Users who make a business journey in another country should generate a website at each moment of the day and copy the files on a web server when they have time and an Internet connection is available.

Robustness or fault tolerance requirements In case of abnormal circumstances, the product shall be unable to function normally. Information that has not been saved shall be lost.

Operational requirements

The operational requirements define the expected physical environment and the expected technological environment. *Expected physical environment* specifies the physical environment in which the product will operate to ensure that the product is fit to be used in its intended environment. *Expected technological environment* specifies the hardware and other devices that make up

the operating environment for the new system.

Expected physical environment The product shall be used in relatively quiet conditions (generally in an office room).

Expected technological environment The product shall be used on a desktop or a laptop equipped with a mouse and a keyboard.

Now we have identified the requirements to develop a conference website, we are able to model the CWPL which is the object of the next section.

3.4 Conference website product line feature diagram

In this section, we build the CWPL on the basis of the information collected previously. To represent the CWPL, we use a FD which is presented in Figure 3.5.

A conference website provides all the useful information about the conference. First, the title, the place, the dates and eventually the logo of the conference are always visible. A conference website also has a collection of pages containing various details of the conference. We can find details such as a description, information about sponsors and submissions, and practical information on the venue and the accommodation during the conference. The *description* page gives the topic of the conference, the *sponsors information* page lists the sponsors of the conference and information about them, the *submissions information* page provides details on how to proceed to send submissions, the *venue* page gives to participant a plan to reach the conference place, the *accommodation* page gives details on the possible lodgings around the conference place. We can also find a *form* allowing to register users on the website. This page requires (*requires* relation) a plugin adding forms.

Second, modules can be added at the content and the core functions of the basic website. A *Newsletter* module allows organizers of the conference to send various information about the conference to the registered users. A *News* module allows to warn users about all novelties of the conference by displaying news on the website. A *Forms* module allows to create customized forms, essentially used to create registration forms. A *Slideshow* module allows to display images on the homepage of the website. A language module allows to select the *default language* of the website. It is also possible to add two other *additional languages* which are different of the default language. Therefore, there are *excludes* relations between the default and the additional languages.

Third, the appearance of the website can be personalized with templates. The look and feel of the website is modified by placing the layouts at different places. The best example is the menu: depending on the chosen template, it can be situated under the header or in a column on the left.

After having analysed the CWPL, we have noticed that it can easily be developed in a CMS. There are several existing CMSs, so a choice must be carried out.

3.5 Selection of a content management system

This section aims to select the most appropriate CMS for our application. First, we present a selection of free CMSs on the market and then we compare them on different criteria. Third, we discuss the choice of a CMS. Finally, we give a more detailed description of the chosen CMS.

3.5.1 Existing CMSs

Making a choice among the quantities of CMSs available on the market [12] is difficult. We have chosen free CMSs distributed under the GNU General Public License (GNU GPL)[1] or GNU Affero General Public License (GNU AGPL)[2]. Our choice is about the 5 following CMSs: PHP-Nuke, SPIP, CMSimple, OpenCMS and WebGUI. We have selected PHP-Nuke and SPIP because we wanted to confront two of the most known CMSs on the market with others. The three others, CMSimple, OpenCMS and WebGUI have been recommended by professional. Those CMSs are presented using seven aspects: description, license, requirements for the installation, administration interface, modules, look and feel, and support (on basis of the information available in April 2009).

PHP-Nuke

Description "PHP-Nuke is a web-based automated news publishing and content management system based on *PHP* and *MySQL*" [70]. The main goal of PHP-Nuke is to help webmasters in the deployment of an automated website allowing users and editors to post news and articles. PHP-Nuke also comes with a *Comments* system allowing users to comment and discuss these articles.

[1] More information about GNU GPL are available on `http://www.gnu.org/copyleft/gpl.html`

[2] On `http://www.gnu.org/licenses/agpl.html`, more details on GNU AGPL are presented.

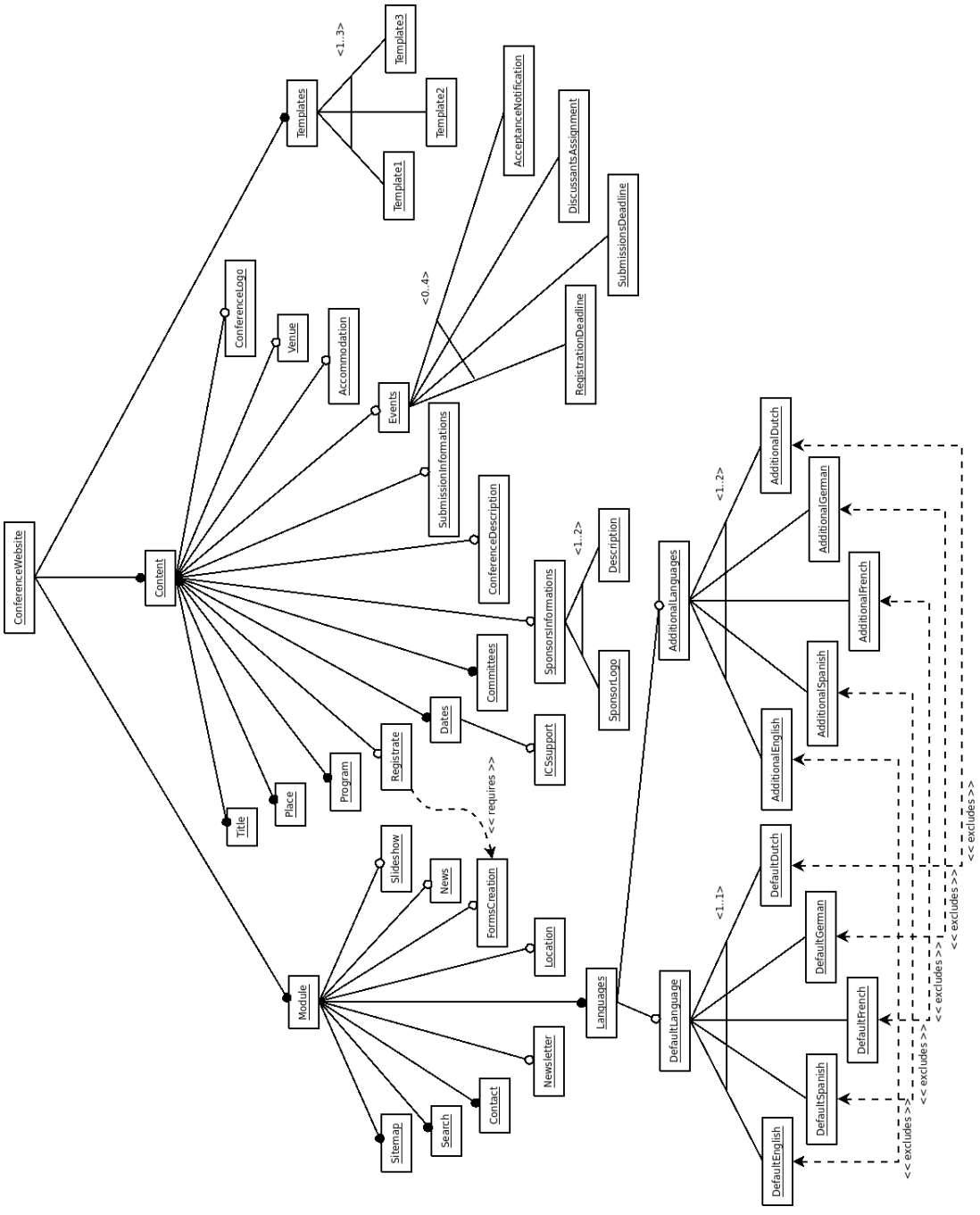


Figure 3.5: Conference website feature diagram

License Starting with version 5.6, the GPL license requires the display of a copyright message on the pages of the website. Last versions of the software were freely downloadable but the latest is the first version for which a download charge is required.

Installation PHP-Nuke requires a web server supporting the *PHP* extension (such as the Apache HTTP Server), as well as an *SQL* database (such as MySQL, mSQL, PostgreSQL, ODBC, ADABAS, Sybase or InterBase).

Administration interface Administrators control all the functionalities of the system using a web-based administration interface.

Modules The standard modules of the PHP-Nuke system are *Downloads*, *Encyclopedia*, *FAQ*, *Forums*, *News*, *Search*, *Statistics*, *Submit News*, *Web Links*, *Journal*, *Members List*, *Private Messaging*. They are part of the PHP-Nuke core but can be extended with additional modules such as an *Internet Forum* or a *Calendar*. PHP-Nuke is also able to support many languages.

Look and feel The *Themes system* of PHP-Nuke allows to customize the look and feel of the template and differentiate it from the standard column layout.

Support The official website of PHP-Nuke [29] contains a lot of helpful information for any user around the world and community websites help users in the support of their system.

SPIP

Description "SPIP is a publishing system for the internet in which great importance is attached to collaborative working, to multilingual environments, and to simplicity of use for web authors" [37]. SPIP consists of a bundle of files, installed on a web server supporting notably multi-user management, articles layout definition without *HTML* usage, easy modification of the structure of the website, etc. "SPIP's benefit consists in managing a magazine type website, i.e. made up mainly of articles and news items inserted in an arborescence of sections nested in each other" [37]. SPIP completely separates the graphic design, the website editorial input (submission of articles and news items) and the website editorial management (organizing sections, validating articles submission, etc.). The need to learn new technical skills and tedious aspects of web publishing are avoided, and thus spared by all stakeholders. A simple interface allows to configure SPIP step by step until the installation is complete and then helps to create the different sections and articles which will appear on the website.

License SPIP is a free and distributed under the GNU GPL license and can thus be used for any type of website (personal or institutional, non-profit

or commercial, etc.).

Installation The use of SPIP requires its installation on a web server with *PHP*, *MySQL* and *FTP* access supports.

Administration interface With its very simple user interface, SPIP allows to set up and update a website using the same application as the one used to browse a website.

Modules A number of varied plugins such as *Agenda*, *Multimedia Reader*, *Accessibility Bar*, *Forums*, etc. are designed for SPIP. Their inclusion and activation is made in a very simple way via in the user interface. Users can also create their own plugins.

Look and feel The look and feel of a SPIP website is managed by templates delivered in *HTML* and *PHP* formats. A large number of templates allowing to personalize a website at low cost also exist.

Support SPIP is developed and used by a community [34] which communicates through a website, a forum [35], a blog [33], mailing lists [36] and meetings.

CMSimple

Description "CMSimple is a simple CMS for the smart maintenance of small commercial or private websites" [8]. It aims to be small, simple and smart. The complete system taking less than 100 KB memory space on a web server makes of it a small CMS. Users just edit the entire website with an *HTML*-editor and upload the content file to have a dynamic website. With no database needed, CMSimple is easy to install and modify as the entire website is saved in a unique *HTML* file. Configuration and language files are also saved in a *.txt* file. A content automatic backup on logout, an integrated online WYSIWYG (What-You-See-Is-What-You-Get) editor with links validation, image handling and online editing of system files make of CMSimple a smart system. CMSimple is not a community portal like SPIP or PHP-Nuke, it is not designed for large volumes of content but to set up quickly a multi-purpose website [9].

License Contrary to other CMSs of this section, CMSimple is released under the Affero General Public license (GNU AGPL). This license is satisfied if a link to the CMSimple Legal Notices [9] is kept visible in the website template. When the link is removed, a license must be purchased.

Installation CMSimple is written in *PHP*, thus it runs on Linux/Apache servers, or on Win32 with Apache or Internet Information Services (IIS).

Administration interface Administrators control all the functionalities of the system using a web-based administration interface.

Modules Like many CMSs, CMSimple has a large variety of plugins optimizing its core functions. Plugins allowing to have a newsletter, FAQ, forum, news and forms systems, are available.

Look and feel The look and feel of a CMSimple website is managed by template files designed by the installer or selected from a set of free templates available through CMSimple. The different templates use *HTML* and *CSS*. Using a default template, a basic website can be configured in a few minutes [9]. Knowledge of *PHP*, *HTML* and *CSS* is required to make a customized template.

Support The free version offers support on a forum [5], a wiki [10] and a mailing-list [7] thanks to a large community of experienced users.

OpenCMS

Description "OpenCMS is a professional, easy to use website CMS helping content managers worldwide to create and maintain beautiful websites fast and efficiently" [25]. The fully browser based user interface includes configurable editors used to structure the content with well defined fields. Alternatively, content can be created through an integrated WYSIWYG editor. No help of externals or professionals is needed with OpenCMS to maintain a public website, an extranet or an intranet. Users can thus focus their attention on the creation of the website content, and publish it in an easy and intuitive way [26].

License Free of licensing costs (GNU GPL), OpenCMS helps to reduce IT costs. As alternative to high expensive and proprietary commercial solutions, OpenCMS provides a professional, cost effective solution ready to be deployed by organizations or enterprises of any size [25]. A full access allows to extend and remodel the source code.

Installation OpenCMS is based on Java and *XML* and can thus be deployed in open source (Linux, Apache, Tomcat, MySQL) or commercial (Windows NT, IIS, BEA Weblogic, Oracle) environments.

Administration interface Once OpenCMS is installed on a web server, administrators can use a web browser to access the user interface of the system from any location.

Modules The user interface of OpenCMS allows convenient bundling of contents or functionalities thanks to an integrated module mechanism. Modules can provide templates to use for website pages, new structured content items, additional frontend functions for a website, for instance to create *photo albums* or interactive *online forms* [26].

Look and feel A sophisticated template engine applies to the whole web-

site the layout in the W3C (World Wide Web Consortium) [44] standard for the compatibility of contents.

Support Support agreements, project and consulting services, and training are proposed by a number of companies around the world. Support is also supplied by an active development community with among other things a wiki [27]. A number of companies provides an optional support, reducing dependency on the deploying software consultancy" [26].

WebGUI

Description "WebGUI is a modular, pluggable, and platform independent content management platform built to give average business users the ability to build and maintain complex websites" [47]. It is not designed to take the time of a busy IT staff, but rather let the content in the hand of the webmaster who just has to focus on the content. It is composed of a *users and groups* system which controls the viewing privileges and the content editing and *versioning* and *workflow* systems. Thanks to these systems, administrators can set up content approval systems to verify if something is published by mistake. They can also set up versioned website content as the content can move through the website. WebGUI is suitable for diverse organizations as it contains a lot of features helping for content creation such as a number of "helper" applications (date, colour pickers) or another application allowing simple and easy to remember URLs assigned to content applications. Style and content are kept separate, hence allowing an easy and quick content management.

License WebGUI is released under GNU GPL license. The code can easily be accessed, changed, and replaced thanks to its modular design.

Installation WebGUI is built as an application framework and allows to easily add new applications for more extensibility and flexibility. With its pluggable macro architecture, it allows developers to create custom applications and functionalities matching company's business processes.

Administration interface The user interface allows to learn how to manage the content and to directly upload images.

Modules WebGUI allows to view the user interface in multiple languages and to display international information such as local date and time. It can also include an *e-commerce* system.

Look and feel To fit with user's skill level, the administration interface allows to manage the available features and the appearance. The template of WebGUI can be customized and keeps the separation between style and website content.

Support The WebGUI community maintains a number of free support resources proposed on the WebGUI website [47]. That website also includes forums [45], a videos library [46] and a wiki [48].

3.5.2 Comparison of selected CMSs

The comparison is based on CMS Matrix [2], a content management comparison tool. It allows to compare up to 10 CMSs at the same time. For each CMS, we confront it using 12 criteria of comparison:

- **Web servers** – Web servers compatible with the CMS.
- **Programming language** – Programming language used to write and/or extend the CMS.
- **Database** – Database engine the CMS uses to store content and settings.
- **Online help** – Availability of an integrated context-sensitive help system.
- **Documentation** – Availability of free or paying documentation.
- **Documentation quality** – Availability of a correct documentation for the administration of the system.
- **Versioning** – Ability to manage levels of system-wide content versioning.
- **Number of administrator accounts** – Number of administrators who can manage the website.
- **Content reuse** – Possibility to mirror content (not copy but reuse) from one location to another on a website.
- **Multi-lingual content** – Support of multiple languages for website creation.
- **Web-based style/Template management** – Availability of a web-based interface for adding styles and templates to the system for design and layout control.
- **Bug resistance** – Sensibility to bugs during upgrade of version.

To judge of the quality of a CMS for our use, we have selected these criteria to test different aspects: ease of install, support, content management, ease of adapting appearance and bug resistance. The ease of install is important to quickly set up a solution at low cost. We are interested in the ease of installation of the CMS on the most common web servers and mainly on free solutions, and the required programming languages avoiding to webmasters the learning of a new language. We also examine the required database because the content is generally saved in a database, which imposes knowledge and memory space on the web server. Both must be minimum. The support allows users to receive help and documentation to manage the system. We focus on the online help that a user can receive for any question, the documentation quality and if it is free or paying. We have also chosen to examine the content management. We highlight the possibility of content versioning to recover it when errors are found and the possibility of reuse a particular content in different pages. The number of administrators which can manage the system is also taken in account. On another aspect, we analyse the ease of adapting the appearance of the website with the possibility of adding multi-lingual content and templates to modify the look and feel. The personalization of the appearance is an important aspect in the choice of a CMS. Finally, we are interested in the bug resistance of the CMS because we cannot consider bugs with the system.

Table 3.18 summarizes the comparison of the different CMSs and is discussed in the next section.

3.5.3 Discussion

We glance through each criterion to identify the main strengths and weaknesses of the CMSs. Table 3.18 summarizes the comparison of the different CMSs according to the defined criteria. With this table, we can highlight the main drawbacks of CMSs in order to select the ideal one. For a better comparison, we give 1 point to a CMS for each criterion when it fits, 0 point otherwise.

On the point of view of the *web servers*, the different CMSs run on Apache. PHP-Nuke, CMSimple and OpenCMS also run on Microsoft Platform (IIS) but this characteristic is unimportant because we are looking for a free solution. As the different CMSs are equals for this criterion, we attribute 1 point to all of them. On the point of view of the *programming languages*, WebGUI has a disadvantage compared to the others. Indeed, Perl is a specific programming language which is less known by webmasters than *PHP* or Java. Moreover, Perl requires a dedicated server. We attribute 0 point to WebGUI and 1 to

Table 3.18: Summary of the CMSs comparison

Criteria	**PHP-Nuke**	**SPIP**	**CMSimple**	**OpenCMS**	**WebGUI**
Web servers	Apache, IIS	Apache	Apache, IIS	Apache, IIS	Apache
Programming language	PHP	PHP	PHP	Java 1.4. +	Perl
Database	MySQL, Postgres, mSQL, Interbase, Sybase	MySQL	No database	Oracle, MySQL, PostgreSQL, MS SQL Server, DB2, AS400 and HSQL	MySQL
Online help	No	No	Yes	Yes	Yes
Documentation	Free	Free	Free	Free	Paying
Documentation quality	Good	Good	Good	Poor	Good
Versioning	No	Yes	Yes	Yes	Yes
Number of administrator accounts	Several	Several	Only 1	Several	Several
Content reuse	No	No	Yes	Yes	Yes
Multi-lingual content	No	Yes	Yes	Yes	Yes
Web-based style/Template management	Limited	Free Add On	Yes	Limited	Yes
Bug resistance	Yes	No	Yes	Yes	Yes

others. On the point of view of the *database*, CMSimple has an advantage over the others: it requires no database. As a conference website is not a complicated website requiring a database, the solution of CMSimple is ideal. Moreover, a website with CMSimple will be less heavy (memory space on the server) than other CMSs. We attribute 1 point to CMSimple and 0 to others.

On the point of view of the *versioning*, only PHP-Nuke keeps no version of the files. The versioning is useful to recover files to a previous version when an error occurs. We attribute 0 point to PHP-Nuke and 1 to others. On the point of view of the *number of administrator accounts*, CMSimple has a disadvantage on the others because only one administrator can manage the website. Other people may manage the website via the single administrator account but this is not a good solution. Contrary to CMSimple, other CMSs have several administration accounts to manage the website. We attribute 0 point to CMSimple and 1 to others.

On the point of view of the *online help*, PHP-Nuke and SPIP have not this functionality. We attribute 0 point to them and 1 to CMSimple, OpenCMS and WebGUI. On the point of view of the *documentation*, at the exception of WebGUI all documentations are free. WebGUI is a free solution developed by a private company which is remunerated by the documentation. We attribute 0 point to WebGUI and 1 to others. On the point of view of the *documentation quality*, OpenCMS has a disadvantage on other CMSs. The documentation available is of poor quality compared to PHP-Nuke, SPIP, CMSimple and WebGUI. Indeed, all documentation sources are incomplete for a correct administration of the system. We attribute 0 point to OpenCMS and 1 to others.

On the point of view of the *content reuse*, PHP-Nuke and SPIP do not offer this functionality. Indeed, for those two CMSs, making a reference to the same document in several pages or reuse the content of a page in another are laborious and complicated tasks. We attribute 1 point to CMSimple, OpenCMS and WebGUI and 0 to the two others. On the point of view of the *multi-lingual content*, only PHP-Nuke does not allow a multi-lingual content. For that reason, one selects only one language. We give 0 point to PHP-Nuke and 1 to the four others.

On the point of view of the *style/template management*, all CMSs allow to personalize the appearance of the website. However, for PHP-Nuke and OpenCMS, the personalization is limited, they have a fixed GUI which is difficult to personalize. We attribute 0 to them and 1 point to the others. On the point of view of the *bug resistance*, only SPIP has a disadvantage. Indeed, the update of SPIP version can cause bugs, which seems to be problematic. We attribute 0 to SPIP and 1 to others.

Table 3.19 summarizes the points attributed to each CMS. As we can see, CMSimple has collected more points than the others. It has no real weakness. However, we can identify a problem of administration: a single administrator is allowed to manage the system. For a conference website, it is not a problem as the amount of pages is not big and, consequently does not require a lot of people to administrate the website. We thus select **CMSimple** for our application.

CMS choice

In this section, we give more details about CMSimple and describe its other characteristics. It is intended for small websites (as conference websites) and the ideal tool for a single user to maintain a website. In addition to these advantages, others features make the strength of CMSimple: a search function, a print version option, a simple mailform, an easy setup of multi language sites, a possible *PHP*-scripting within the content (named CMSimple Scripting), a possible integration with 3rd party scripts (using CMSimple scripting), and an automatic backup on logout and the respect of *XHTML* standard. The core is designed to generate *XHTML* code, the external editors also respecting this encoding (as TinyMCE [40]). The template can easily be adapted to be entirely compatible with *XHTML/CSS*. CMSimple has a complete administration interface in which different plugins can be installed.

Table 3.19: Summary of the points attributed to selected CMSs

Criteria	**PHP-Nuke**	**SPIP**	**CMSimple**	**OpenCMS**	**WebGUI**
Web servers	1	1	1	1	1
Programming language	1	1	1	1	0
Database	0	0	1	0	0
Versioning	0	1	1	1	1
Number of administrator accounts	1	1	0	1	1
Online help	0	0	1	1	1
Documentation	1	1	1	1	0
Documentation quality	1	1	1	0	1
Content reuse	0	0	1	1	1
Multi-lingual content	0	1	1	1	1
Web-based style/Template management	0	1	1	0	1
Bug resistance	1	0	1	1	1
Total	6	8	11	9	9

Chapter 4

Approaches to configuration

In this chapter, we develop two strategies to automate the creation of a conference website from a product line. The first strategy is an application specific approach which consists in the development of a wizard generating a conference website. The second one is a generic approach which uses a variability modelling and configuration tool to generate the conference website. The two approaches will then be compared and their limitations highlighted.

4.1 Application specific: wizard

This section presents the application specific solution developed during a traineeship in the University of Seville. We have chosen to implement it as a wizard, which is a good solution to complete the requirements that we have underlined, as we will show. A wizard is a user interface that guides the user through a potentially complex task with a sequence of dialog boxes [74]. These dialog boxes lead the user step-by-step by asking questions or telling what data to enter and moving forward and backward through, filling in the required details [49]. Wizards are often used for complex or infrequent tasks in which the user is unfamiliar with the steps involved. [39].

We have decided, in this section, to not speak about the architecture because a software architecture describes the different components of a system and the relations between them. As our wizard contains a unique component, the generation algorithm, it can be decomposed with difficulty in a precise architecture. It is composed of a GUI which allows users to select desired features and an algorithm which generates the CMS with the correct configuration.

First, we develop the structure of the wizard which details how it is organized. Then, we describe its GUI.

4.1.1 Structure

To build the structure of the wizard, we start from the FD presented previously (see Figure 3.5 in Section 3.4) to translate it into a simple easy-to-use graphical application. To define the structure, we use two notations described in the Unified Modelling Language [41] (UML): Class Diagrams (CDs) and Sequence Diagrams (SDs). First, we build the CD on basis of the FD. Then we define SDs to make clear how the wizard works. Finally, we give a solution for each functional requirement defined in Section 3.3.5. We focus only on the functional requirements because they have a major influence on the design of the solution rather than the non-functional requirements which have a minor influence. Some of the latter, are automatically taken in account in the development of an application such as the ease of use, usability or performance requirements.

Class diagram

In this section, we describe the CD representing our system. CDs are used to describe the structure of a system within a model by showing the system's classes. In an object-oriented application, classes have their attributes (member variables), operations (member functions) and relationships with other classes [76].

Our CD, derived from the FD described in Section 3.4, represents a conference website. When an instance of it is created, the generation can be executed. As shown in Figure 4.1, our CD is composed of the six following classes: *ConferenceWebsite* (main class), *Dates*, *Event*, *Page*, *Module* and *Template*. A conference website (*ConferenceWebsite* class) is characterized by six attributes: *title*, *place*, *logo*, *default language*, list of *additional languages*, and *installation files*. Each conference website is identified by the title of the conference. A conference website has at least one date (*Dates* class). Each date is identified by its *begin* and *end dates*. In addition to the conference dates, other dates, named events (*Event* class), can be added. An event possesses a *name* which identifies it. A conference website can be composed of different pages (*Page* class) which have a *title* and *content*. A page has its own title. It can own modules (*Module* class) to complete different functions such as manage a newsletter, forms, news, etc. Each module has a unique *name* and *installation files* which must be inserted in the website installation files. Finally, the website possesses at least one template (*Template* class) to modify its appearance. It has a default template and other can be added. Like modules, each template has a unique *name* and *installation files*.

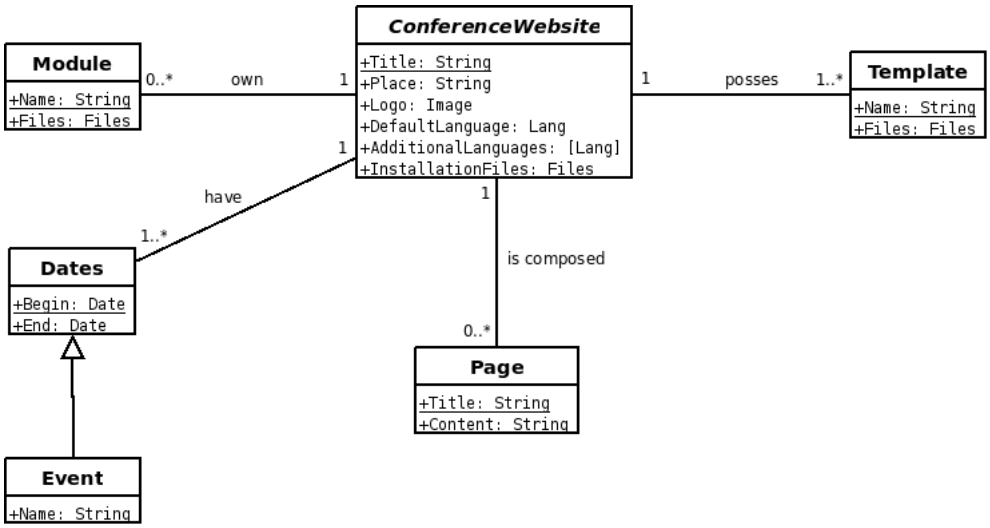


Figure 4.1: Class diagram of the conference website

Sequence diagrams

Here, we describe how the information is processed by the wizard through SDs which show how processes operate with one another and in which order.

SDs are used to represent or model the flow of messages, events and actions between the objects or components of a system [85]. At the top of the diagram are displayed the header elements representing objects or classes. The vertical direction called lifeline, representing the time, shows the sequence of interactions between them. These interactions are displayed horizontally. Static diagrams or specifications make difficult the extraction of the system behaviour. SDs are useful design tools which solve this problem by providing a dynamic view of it. Primarily used to design, document and validate the architecture, interfaces and logic of the system, SDs describe the sequence of actions needed to be performed to complete a task or scenario.

Figure 4.2 presents the sequence of actions to carry out the conference website generation. Three objects operate here: *user*, *system*, *calendar* and *CMS*. The *user* is the person using the system. The *calendar* allows to select a specified date on a graphical interface. The *system* is the application generating the conference website. The *CMS* website is the system which will be generated. The user begins by selecting the general information like the title and the place of the conference. Optionally, (s)he can add a logo on the website template by specifying the file directory. Then (s)he selects the

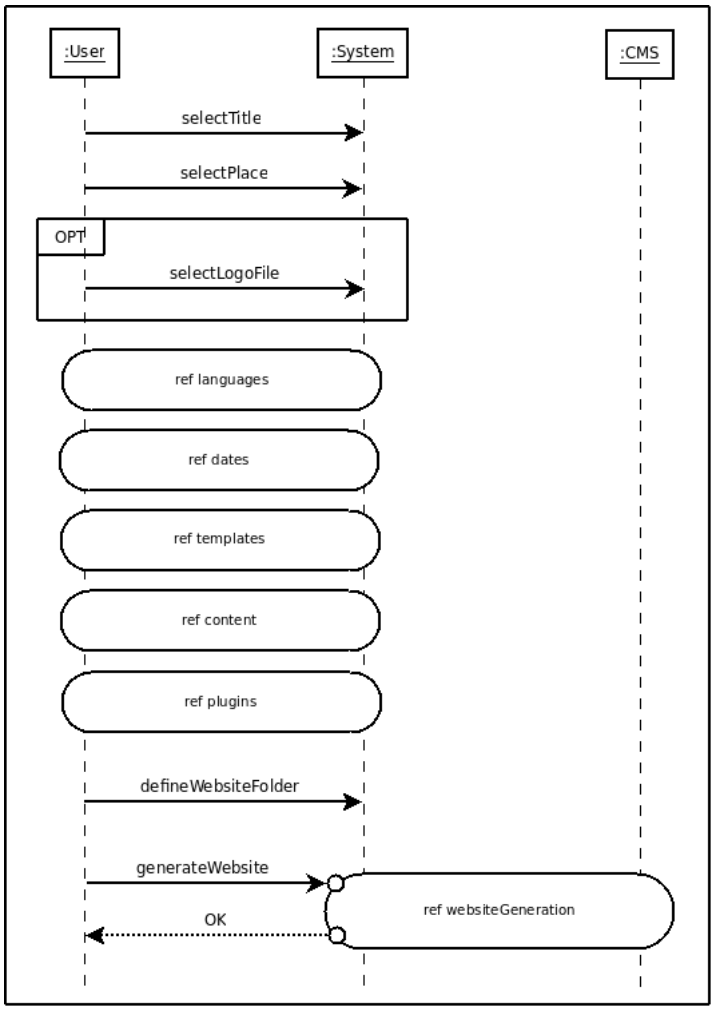


Figure 4.2: Application specific approach: general sequence diagram

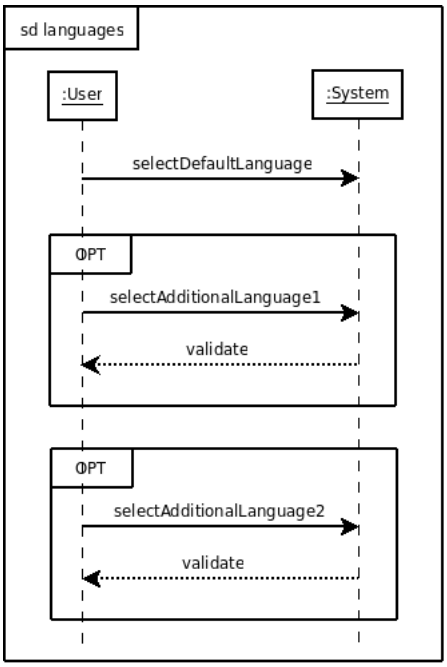


Figure 4.3: Application specific approach: languages sequence diagram

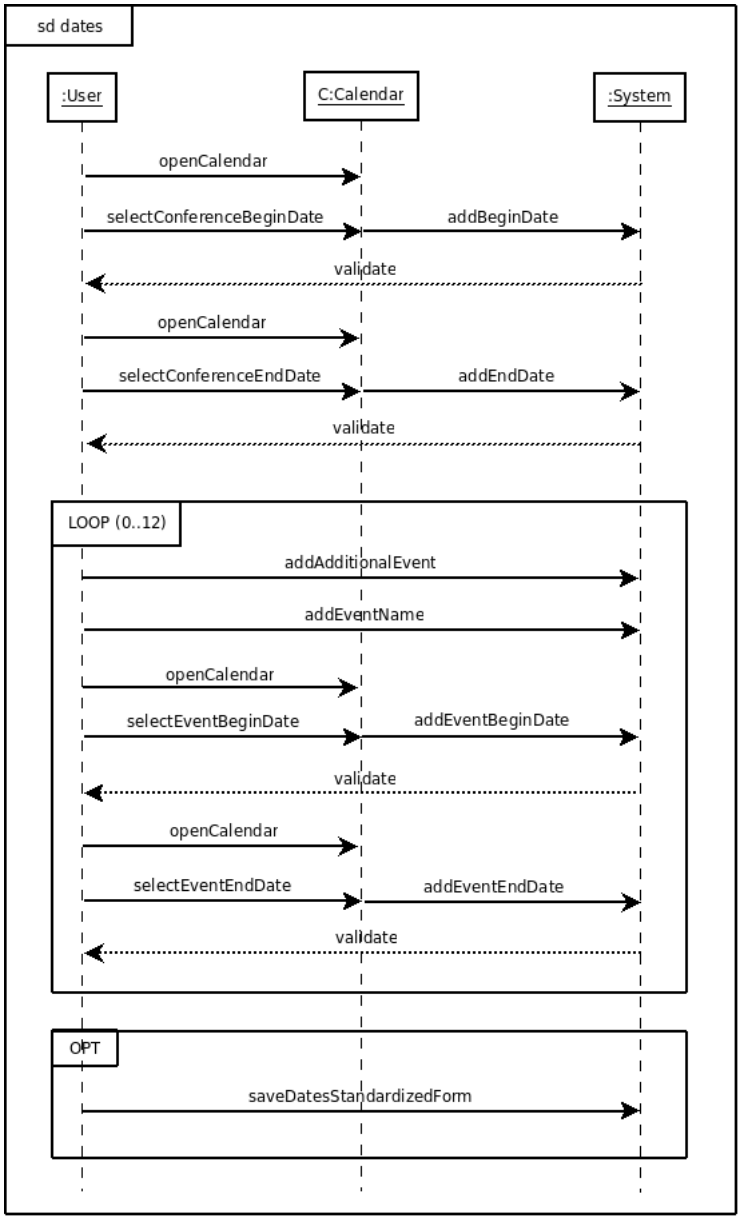


Figure 4.4: Application specific approach: dates sequence diagram

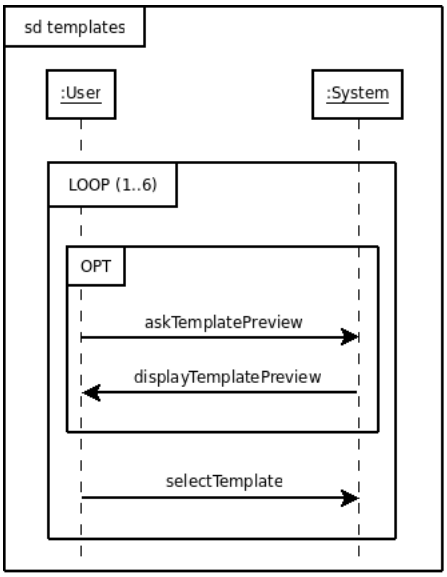


Figure 4.5: Application specific approach: templates sequence diagram

languages and dates, respectively described in Figure 4.3 and Figure 4.4. For the languages, (s)he first selects the default language before adding if he wants one or two additional languages. At each addition, the system verifies that the user does not select a language already selected. For the dates, (s)he starts by defining the begin and end dates of the conference. For that, a *calendar* is opened to easily choose a date. For the two dates, the system validates them by checking if they are not in the past or if the end date is not before the begin date. After the selection of the conference dates, (s)he can add up to 12 events by specifying the event name and the begin and end dates (via the calendar). Here, the dates are also checked by the system. Once all the dates have been defined, the user can choose to save them in a standardized form (which uses a particular file) to be imported then in a calendar. As described in Figure 4.5, (s)he is then invited to choose the templates which will allow to personalize the website. (S)he can select up to six templates and display a preview of each of them to have a better idea of the appearance. The next operation is to define the content pages of the website. As shown in Figure 4.6, the user can, if (s)he wants, add different pages to give information about the sponsors, registration, submissions, accommodation, venue, program, committee and provide a conference description. The last operation of selection is devoted to plugins as described in Figure 4.7. The user can select a newsletter, news, forms and

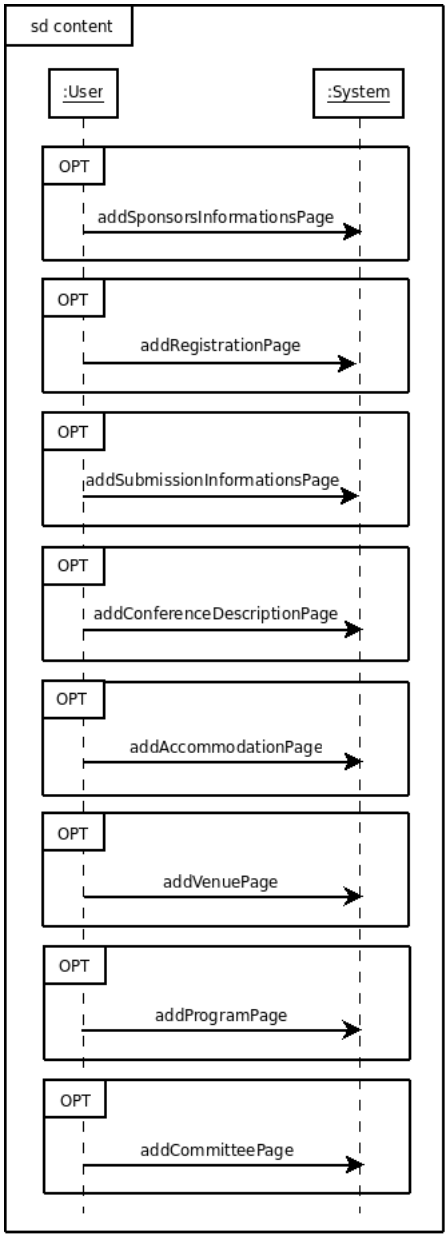


Figure 4.6: Application specific approach: content sequence diagram

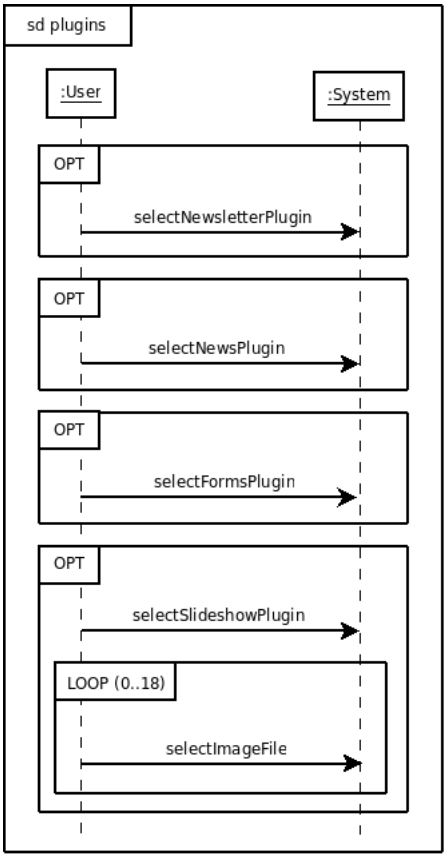


Figure 4.7: Application specific approach: plugins sequence diagram

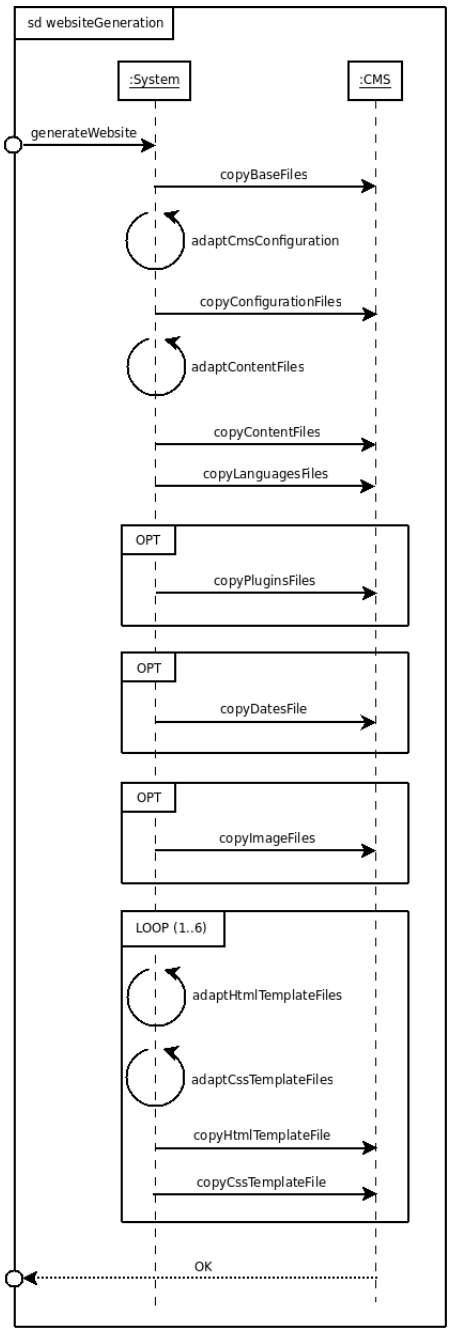


Figure 4.8: Application specific approach: generation sequence diagram

slideshow plugin. If (s)he selects the slideshow plugin, (s)he can add up to 18 images by specifying the files directories. Before generating the website, the user must define in which folder copy the website files. (S)he can then launch the generation. As detailed in Figure 4.8, during the generation, the system begins by copying the basis files of the CMS which must not be modified. Then it adapts the configuration and content files and copies them as well as the languages files in folder of destination. After that the system copies plugins files, dates file (in standardized format) and images for the slideshow depending on the selection made by the user. Finally, the system adapts the templates files (*HTML* and *CSS* files) that the user has chosen to possibly add the logo and the news plugin and copy files. The website is then ready to use.

Implementation choices

In this section, we detail the implementation choices made for our application. We have chosen to implement it in the Java programming language because Java provides all the necessary components to build a GUI thanks to the *Swing* library [22]. *Swing* is a widget toolkit for Java which allows to create a graphical interface which will be identical on all the platforms. That allows to respect the first constraint made in the requirement engineering (see Section 3.3.4) which imposes to the system to run on different operating systems. Another reason to use Java is that we validate the configurations with FaMa-FW [14]. As we have seen in Section 2.3.1, FaMa-FW is an Eclipse plugin written in Java which will be easily integrated in our application. We use FaMa-FW because it has been developed at the University of Seville where we have elaborated this wizard.

Solutions to requirements

In this section, we present a solution which seems to be the most appropriate according to our experience for each functional requirement. We use the functionalities of Java and CMSimple.

Requirement 1 The first requirement (defined in Table 3.5), which consists in the validation of each configuration of conference website, is solved thanks to FaMa-FW and the GUI especially designed to avoid errors. Indeed, we have some features which must be completed by a textual entry (for instance, title of the conference) or others which require or exclude other features. For example, the English feature of the default language excludes the English feature of the additional languages. It is the same for the other languages and this allows to do not have two times the same language. For the first type of relationship,

the required feature is automatically selected even if the user has not chosen it. For the second one, a feature excluded by another one does not appear on the interface to avoid its selection. To avoid these types of errors, the GUI manages permanently the display to avoid the selection of two identical languages or imposes the completion of the mandatory information. In addition, FaMa-FW completes the GUI in order to validate the configuration. Indeed, when the configuration is finished, FaMa-FW checks the configuration. Normally, FaMa-FW does not identify errors because the GUI avoids them but it is above all useful for a future evolution of the wizard allowing to check manually at each moment the configurations. Once the configuration is validated, the website generation can be completed.

Requirement 2 This requirement (defined in Table 3.6), which imposes to offer the possibility to display the conference logo on the templates of the website, is taken in account by the use of the *JButton* and *JFileChooser* components of the *Swing* library. *JButton* is an implementation of a "push" button which allows to execute an action when it is pressed. *JFileChooser* provides a simple mechanism for the user to choose a file. Those two components, placed on the interface, allow to select the conference logo file. When the button is pressed, the filechooser opens and the user can navigate in his/her personal folders and files to select the desired file. Once the configuration is completed, the file is uploaded in the configuration files of the CMS.

Requirement 3 This requirement (defined in Table 3.7), which offers to choose 3 languages among 5 languages as website languages (one by default and 2 additional), is solved by the use of *JCombobox* components placed on the interface allowing to select the languages. A *JCombobox* is a component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request. We have placed 3 comboboxes for the 3 languages. Once the default language is chosen, this choice is removed of the list of the second and the third combobox to avoid to select two times the same language. The same operation is executed for the third combobox after the choice of the second one. We have chosen to propose the five following languages: English, Spanish, French, Dutch and German.

Requirement 4 This requirement (defined in Table 3.8), which consists in offering the possibility to add a minimum of 6 events in addition to the conference dates, is solved by the use of *JPanel* and *JTextField* components of the *Swing* library. A *JPanel* is a generic lightweight container which is used to contain other components. A *JTextField* is a lightweight component that

allows the editing of a single line of text. We have especially designed a panel to add up to 12 events with their details: event name and begin and end dates. This panel contains, for the 12 events, three textfields to insert the name and the two dates.

Requirement 5 This requirement (defined in Table 3.9), which proposes a calendar to easily choose dates or events of the conference, is taken in account by appealing to the *JCalendar* library [23]. *JCalendar* is a Java date chooser bean for graphically picking a date. It is composed of several other Java beans, a JDayChooser, a JMonthChooser and a JYearChooser. All these beans have a locale (specific geographical region) property, provide several icons and their own locale property editor. Also part of the package are a JDateChooser, a bean composed of an IDateEditor (for direct date editing) and a button for opening a JCalendar for selecting the date. In our application we use the JDateChooser. For each selection of a date, a button is placed and when it is pressed, the calendar opens.

Requirement 6 This requirement (defined in Table 3.10), which consists in offering the possibility to save all dates in a standardized format, is solved by the use of the iCalendar standard [20]. The iCalendar standard uses the *ics* extension to point out a file containing information about a calendar. The *ics* extension allows to share, send by email and import dates in a calendar. At the generation of the website, the conference information and added events information (event name and dates) are used to compose the *ics* file. Table 4.1 describes the fields that we have used to compose our file. Here we define the fields that we needed to insert our information (see [20] to consult a description of all fields). The X-WR-CALNAME field is the title of the conference. For each event, we have a BEGIN:VEVENT field, followed with information about the event and closed by an END:VEVENT field. As information about the event, we have the begin and end date (DTSTART;VALUE=DATE: date and DTEND;VALUE=DATE: date), the event name (UID: name) and the date of creation of the event (CREATED: date of creation)

Requirement 7 This requirement (defined in Table 3.11), which consists in offering the possibility of adding a plugin managing a newsletter, is taken in account by adding the *Geniz newsletter* plugin [6] of CMSimple. The plugin files are placed in the configuration files of CMSimple and users access it by the administration interface.

Table 4.1: Example of *ics* file content

```
BEGIN:VCALENDAR
PRODID:-//Google Inc//Google Calendar 70.9054//EN
VERSION:2.0
CALSCALE:GREGORIAN
METHOD:PUBLISH
X-WR-CALNAME: + title of the conference
X-WR-TIMEZONE:Europe/Madrid
BEGIN:VEVENT
DTSTART;VALUE=DATE: begin date
DTEND;VALUE=DATE: end date
DTSTAMP:20090119T091850Z
UID: event name
CLASS:PUBLIC
CREATED: date of creation
LAST-MODIFIED: date of creation
SEQUENCE:0
STATUS:CONFIRMED
SUMMARY:
TRANSP:TRANSPARENT
END:VEVENT
END:VCALENDAR
```

Requirement 8 This requirement (defined in Table 3.12), which imposes to offer the possibility of adding a plugin managing news related to the conference, is solved by the use of the *Advanced news* plugin [4] proposed on the CMSimple website. As all CMSimple plugins, users administrate the plugin via the administration interface of the CMS. Its files are inserted with the configuration files of CMSimple.

Requirement 9 This requirement (defined in Table 3.13), which consists in offering the possibility to add a plugin managing forms, is solved by adding the CMSimple *Advanced form* plugin [3]. In the administration interface of CMSimple, an area devoted to the installed plugins allows to manage the forms plugin. Its files are placed with other plugin files.

Requirement 10 This requirement (defined in Table 3.14) imposing to offer the possibility of adding a plugin displaying a slideshow on the homepage, is

taken in account by adding the *PHP/SWF Slideshow* script [30]. This script is placed in the configuration of CMSimple and a simple command placed in the content file allows to display the slideshow on the homepage. The generator allows to select images to add into the slideshow.

Requirement 11 This requirement (defined in Table 3.15), which consists in offering the possibility to add up to 10 images to the slideshow, is solved by the use of a *JPanel* component of the *Swing* library. This panel, being a container, is used to contain other components. We have especially designed it to add up to 18 images with, for each of them, a *JTextField* (for the edition of a single line of text) and a *JFileChooser* (for the user to choose a file). Once the file is selected with the filechooser, its name is placed in the textfield. At the generation, the files are uploaded in the CMSimple images folder and displayed in the slideshow.

Requirement 12 This requirement (defined in Table 3.16) imposing to offer the possibility to add 5 other templates in addition to the default one, is solved by the use of *JPanel* and *JButton* components. The panel especially designed to present 5 templates in addition to the default one possesses a button for each template with an image representing it. When a user clicks on the template's image, a new window opens with this image in wide size.

4.1.2 Graphical user interface

In this section, we define how the GUI of the wizard has been designed. It is composed of 8 panels which allow to generate a conference website.

When users launch the wizard, they are welcomed with the first panel (shown in Figure 4.9) which invites them to add general information about the conference. They are invited to add the title and the place, if necessary upload the conference logo, and select the default language as well as 2 other optional languages thanks to three comboboxes as we have seen previously (see requirement 3 in Section 3.3.5). Users can navigate to the next panel with the *Next* button and return in the previous to make a change with the *Previous* button excepted for the first panel.

After having added the general information, users are guided to the second panel (shown in Figure 4.10) allowing them to define the begin and end dates of the conference. To specify the dates, users have to click on the calendar button to open it (see Figure 4.11 to have a view of the calendar). It also allows to add additional events. If the combobox *Add additional dates* is selected, they are led to another panel allowing them to add up to 12 events with their name

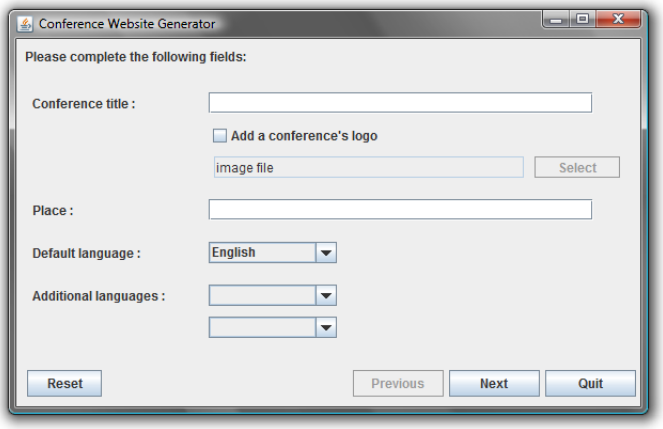


Figure 4.9: Application specific approach: screenshot of the start panel

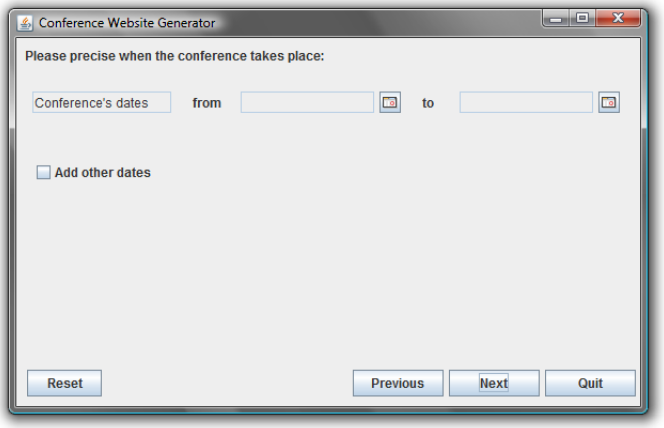


Figure 4.10: Application specific approach: screenshot of the conference dates choice panel

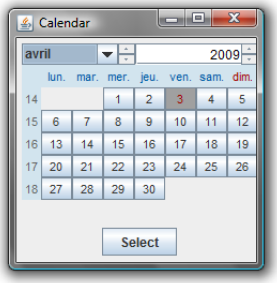


Figure 4.11: Application specific approach: screenshot of the calendar window

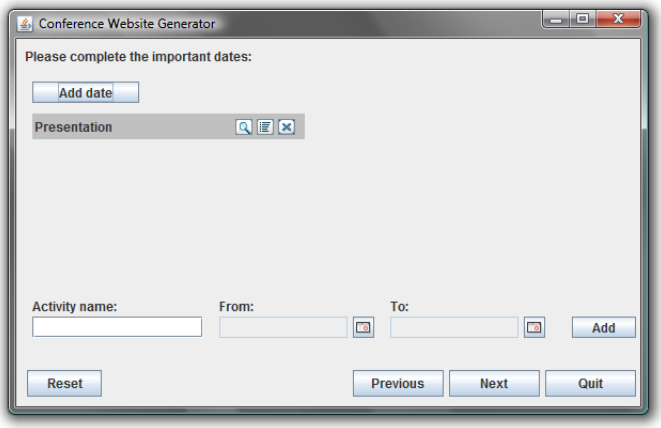


Figure 4.12: Application specific approach: screenshot of the events choice panel

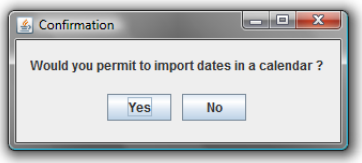


Figure 4.13: Application specific approach: screenshot of the date confirmation window

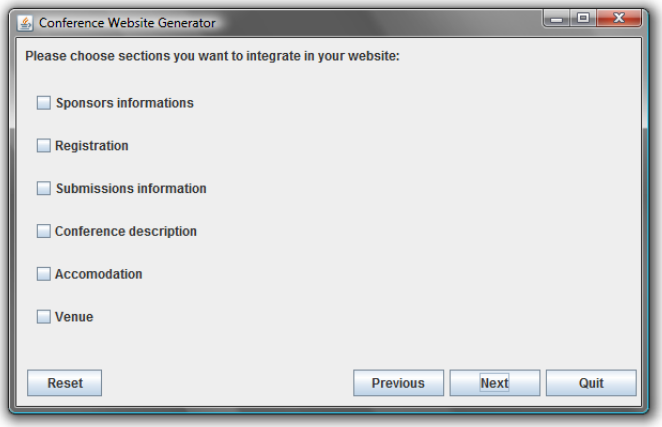


Figure 4.14: Application specific approach: screenshot of the content choice panel

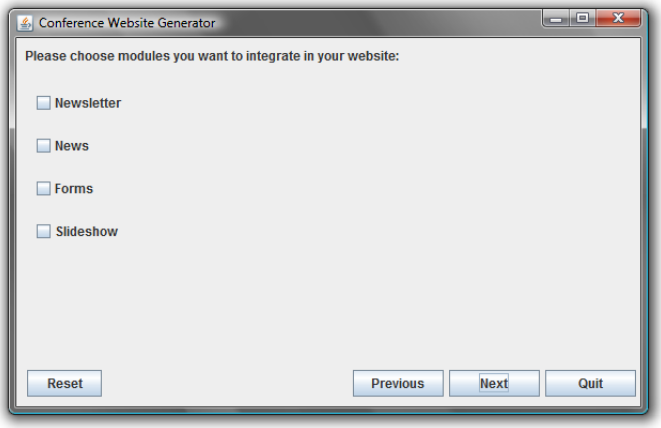


Figure 4.15: Application specific approach: screenshot of the modules choice panel

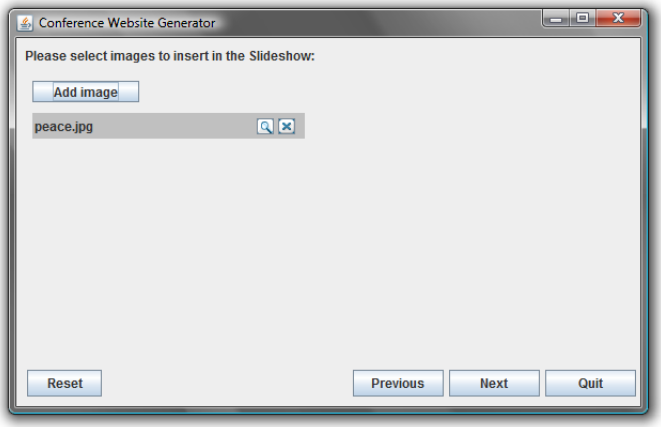


Figure 4.16: Application specific approach: screenshot of the images addition panel

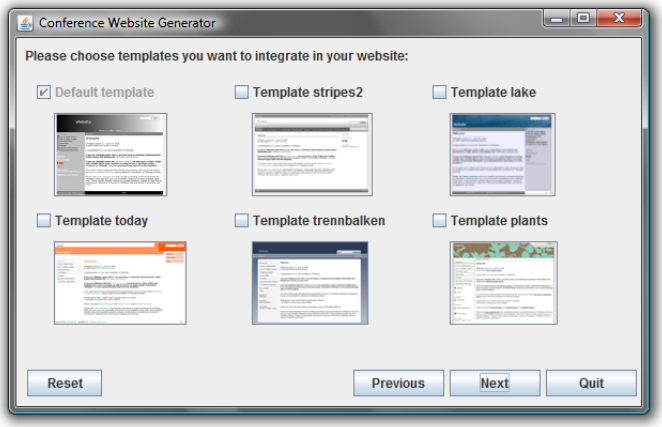


Figure 4.17: Application specific approach: screenshot of the templates choice panel

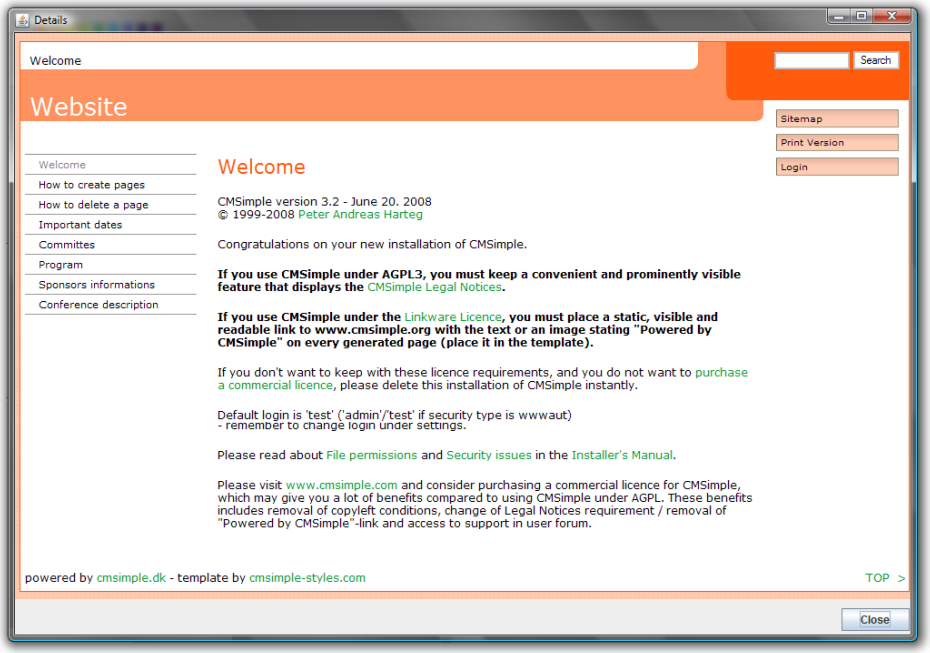


Figure 4.18: Application specific approach: screenshot of the template details window

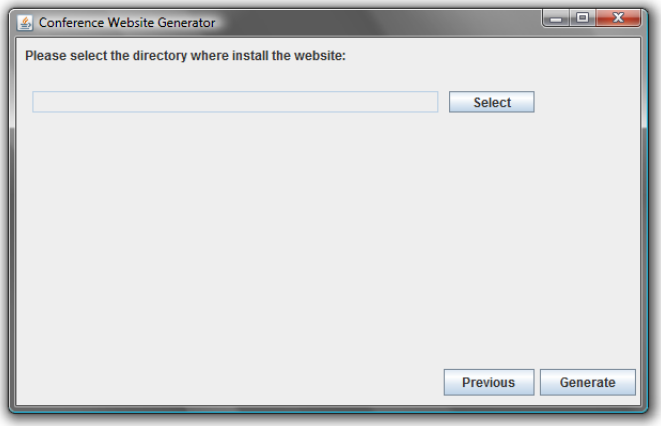


Figure 4.19: Application specific approach: screenshot of the generation panel

and associated begin and end dates (see Figure 4.12). Once added, users can see events details, modify or delete them thanks to the three buttons (visible in the Figure 4.12, see the example of the event "Presentation"). When the user clicks on the *Next* button, a popup (visible in Figure 4.13) asks if (s)he wants to save dates in an *ics* file.

The third panel (visible in Figure 4.14) allows users to select desired content pages. They have the choice to add six content pages: sponsors information, registration, submission information, conference description, accommodation and venue.

As shown in Figure 4.15, the fourth panel allows to select modules to add on the website. Users have to choice to add the newsletter, news, forms and slideshow plugins. If the slideshow module is selected, they are led to an additional panel (see Figure 4.16) to select images to insert in the slideshow. Users can add up to 18 images and for each, see details or delete it.

The fifth panel, shown in Figure 4.17, is designed to select templates. The website is composed of a default template which is selected by default, but users can select 5 other templates to change the appearance of the website. When they push a template image, a new page (visible in Figure 4.18) opens with the selected template in large size to have a better view of it.

The sixth panel, presented in Figure 4.19, is the last before the generation. It allows to select the directory where the files of the website have to be copied. When users click on the *Generate* button, the generation is launched and users are warned when it is finished.

4.2 Generic: feature-model based

After having developed a first approach, this section presents another approach: a generic solution. This second solution is feature-model based, it means that it is purely FD-based in using a FD to make the choice of features. We envisage the use of an external application to help us to reach our aim: a variability modelling and configuration tool. As we have seen in Chapter 2 Section 2.3, there are different variability modelling and configuration tools, but after having compared a selection of them, we have chosen Pure::variants (see Section 2.3.2 for the details of the comparison). Pure::variants allows to build a FD and then display it in different formats. The user selects the desired features directly on the diagram and then a generator takes the configured FD (with the selected features) exported by Pure::variants. The generator translates the configured FD in a ready-to-use website.

As in the previous section, we have made the choice to not speak about architecture because we have just developed the algorithm of generation. Our

solution uses Pure::variants and a generator (which contains the algorithm of generation) with a simple working. Pure::variants has a complex architecture that we do not detail here for brevity. The aim of the generator is simply to check which features are selected and copy the associated files.

We develop first the structure of the feature-model based application. Then we explore its GUI.

4.2.1 Structure

As for the application specific solution, we start from the FD presented in Section 3.4 to build the structure of the feature-model based solution. Our solution is composed of two parts which participate to build a functional website. To present the structure, we use the SD [42] notation described in UML [41]. First, we define SDs to present clearly how our solution operates. Then we provide a solution for each functional requirement defined in the Section 3.3.5. As for the application specific approach, we focus only on the functional requirements. Finally we describe the technical complications that we have encountered in the development of this solution.

Sequence diagrams

In this section, we describe the working of the feature-model based solution through SDs which show how processes operate with one another and in which order. As we have already said, SDs are used to represent or model the flow of messages, events and actions between the objects or components of a system [85]. They describe the sequence of actions needed to be performed to complete a task or scenario.

Figure 4.20 presents the sequence of actions needed to generate a conference website in which five objects intervene: *user*, *Pure::variants*, *Feature Diagram FD*, *system* and *CMS*. The *user* is the person using Pure::variants and the system. *Pure::variants* is a variability modelling and configuration tool that allows to configure and edit a FD. The *Feature diagram FD* is the FD created by Pure::variants. The *system* is the application generating a conference website on basis of the FD. The *CMS* is the system which will be generated. The user begins the selection of the different features using Pure::variants (as shown in Figure 4.21) to configure a FD which will be used later by the system. First, (s)he selects the title, the place of the conference and possibly a logo by specifying the file directory. (S)he continues by selecting the default language and, if (s)he wants up to two additional languages. At each selection, Pure::variants checks that the configuration is valid. It avoids that the user selects a language previously chosen. Then (s)he selects the

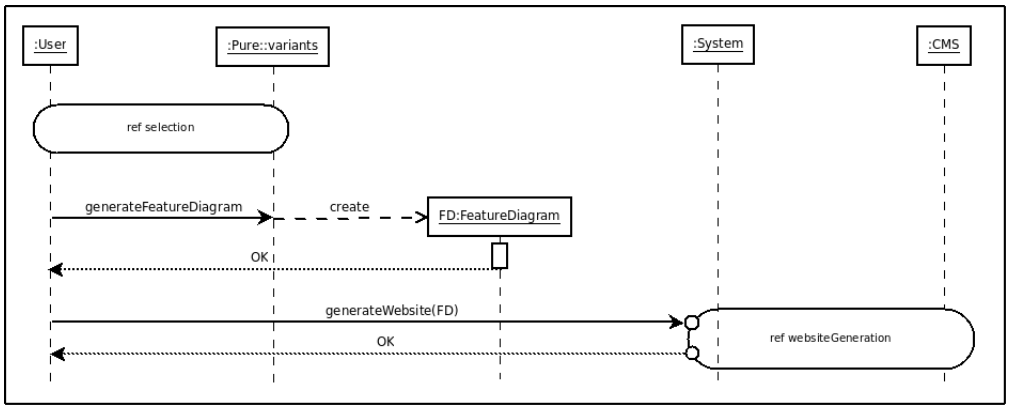


Figure 4.20: Generic approach: general sequence diagram

dates, as shown in Figure 4.22. In addition, (s)he can add up to 12 events, either in selecting an existing or in adding new events in the FD. Adding more than 12 events is also possible. For both, the user has to define the event name, and the begin and end dates. The user can also save all the inserted dates in a standardized form (in the dates file) by selecting the related feature. After having configured the dates, (s)he is then invited to select the desired templates among the six proposed. The next operation is to define the content pages and the plugins of the website. As shown in Figure 4.23, the user can, if (s)he wants, select the different pages that (s)he wants to make appear on the website to give information on the sponsors, registration, submissions, accommodation, venue, program, committee and provide a conference description. Then, (s)he can select the plugins as described in Figure 4.24. The user can choose the newsletter, news, forms and slideshow plugins. If (s)he has made the choice of the slideshow plugin, (s)he can specify a list of up to 18 attributes (here images) by giving their file directory. In Pure::variants, it is possible to add more images. Before generating the website, the user finishes the operation of selection by specifying the folder in which copy the website files. Once the configuration is finished, (s)he has to use the FD export function of Pure::variants to generate the configured FD. Then (s)he provides it in a generator as input to generate the website. As it is detailed in Figure 4.25, during the generation, the system copies the basis files, the configuration and content files (after having adapted them in function of the selected features of the FD), and the languages files in the destination folder. After that, it copies the selected plugins files, the dates file (in standardized form) and the images

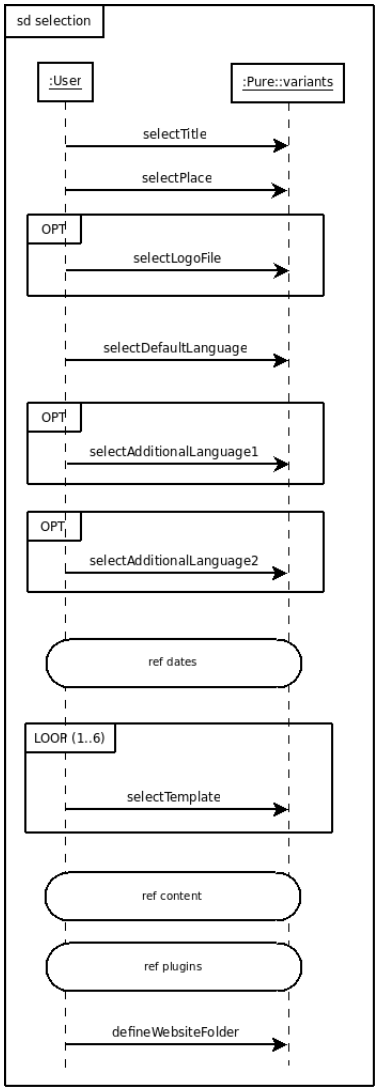


Figure 4.21: Generic approach: selection sequence diagram

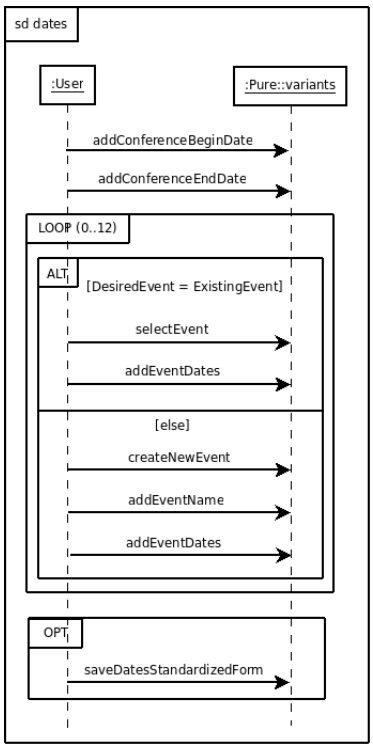


Figure 4.22: Generic approach: dates sequence diagram

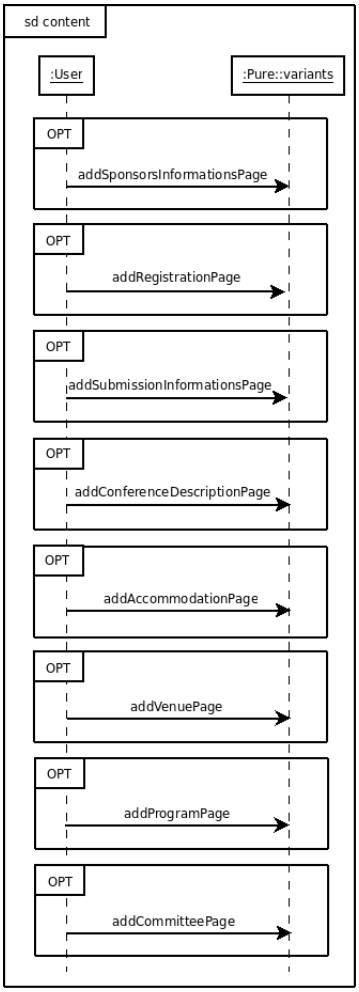


Figure 4.23: Generic approach: content sequence diagram

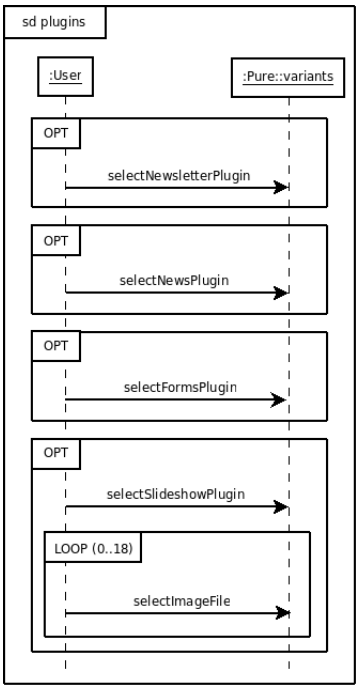


Figure 4.24: Generic approach: plugins sequence diagram

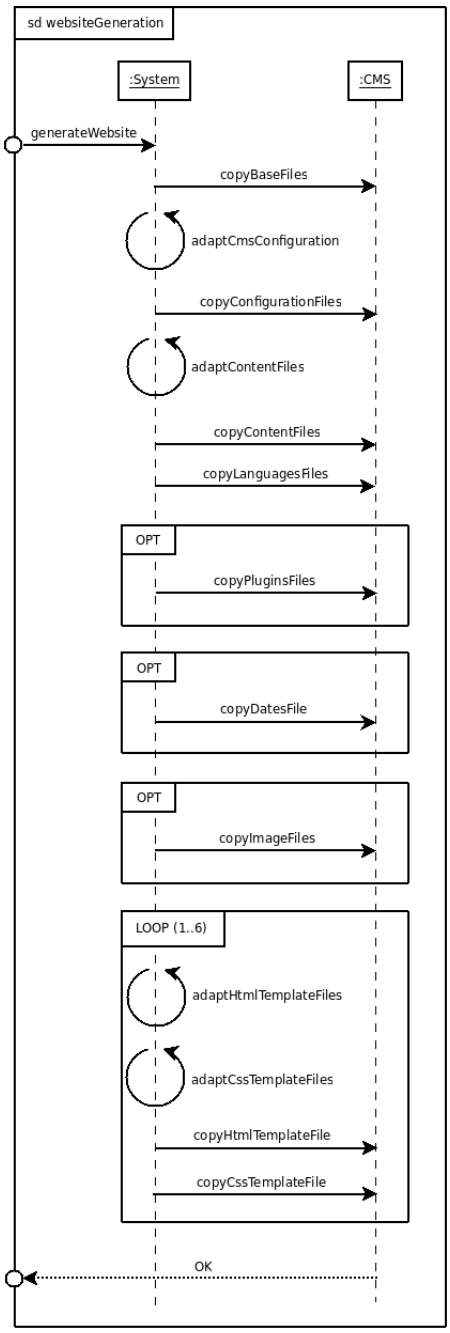


Figure 4.25: Generic approach: generation sequence diagram

for the slideshow. Finally, it adapts and copies the templates files (*HTML* and *CSS* files) that the user has chosen. The website is then ready to use.

Implementation choices

Here we describe the implementation choices made for our solution. As detailed in the SDs, we use two tools: Pure::variants and a generator. Pure::variants is used to define features of the conference website that we desire. Once the configuration is completed, the configured FD is translated into *XML*. The generator takes the configured FD generated by Pure::variants as input and translates it into a functional website. Our solution is divided in two tools due to integration problems of scripts in Pure::variants. Indeed, Pure::variants allows to add scripts to execute different actions but in our case, it was not possible. So we have decided to create another tool: the generator.

The generator has been implemented in the Java programming language for the simple reason that generation and files copy algorithms developed for the application specific approach (see Section 4.1) could be reused. We just had to develop a parser to parse all the features of the FD.

Solutions to requirements

In this section, we provide a solution which seems to be the most appropriate according to our experience for each functional requirement. We use the functionalities of Pure::variants and CMSimple.

Requirement 1 The first requirement (defined in Table 3.5), which consists in the validation of each configuration of conference website, is automatically solved in Pure::variants. Indeed, it verifies the configuration at each moment to avoid conflicts. Mandatory features are automatically selected and Pure::variants prevents their deselection. In the same way, when a feature which requires another is selected, the required feature is automatically selected and for the exclusion, it deselects the features excluded by others. In addition, Pure::variants allows to make a manual validation at each moment.

Requirement 2 This requirement (defined in Table 3.6), which imposes to offer the possibility of displaying the conference logo on the templates of the website, is completed with Pure::variants. It offers the possibility to add an attribute or a list of attributes for each feature. An attribute has a type which is specified by the user. Here we add an attribute of "Directory" type which contains the directory of the logo image. Then the parser of the generator takes it to upload the file in the CMS files.

Requirement 3 This requirement (defined in Table 3.7), which offers to choose 3 languages among 5 languages as website languages (one by default and 2 additional), is solved thanks to the *Exclude* and *Cardinality-based* relationships of FD (these concepts are presented in Chapter 2 Sections 2.2.1 and 2.2.2). Indeed, the cardinality of the default language is defined to only one language and the default and additional languages exclude each other. It allows thus to have only one language by default and a maximum of two others which are different. As we have seen in the first solution to requirements, the FD is permanently verified by Pure::variants, so errors are avoided.

Requirement 4 This requirement (defined in Table 3.8), which consists in offering the possibility of adding a minimum of 6 events in addition to the conference dates, is solved in Pure::variants. This tool allows to add easily new features, so users can add 6 or more event features with 3 attributes, one for the event name, one for the begin date and one for the end date.

Requirement 5 This requirement (defined in Table 3.9), which proposes a calendar to easily choose dates or events of the conference, cannot be solved because, during the feature selection in Pure::variants, it is not possible to display a calendar in a graphical interface. The use of different formats of dates makes difficult their verification, so this requirement is useful to avoid this problem. Nevertheless, with the definition of the types of attributes in Pure::variants, we can specify the "date" format and thus users must respect this format. The validity of the dates can be verified in Pure::variants thanks to a script. As the customer dissatisfaction for this requirement has a low level, it is not a real problem if it is not totally respected.

Requirement 6 This requirement (defined in Table 3.10), which consists in offering the possibility to save all dates in a standardized format, is simply solved with the FD and the iCalendar standard [20]. The *ics* file allows to share, send by email and import dates in a calendar. When the *SaveIcsFile* feature is selected, the generator takes all the dates and conference information, and generates the *ics* file.

Requirement 7 This requirement (defined in Table 3.11), which consists in offering the possibility of adding a plugin managing a newsletter, is taken in account in the same way that with the wizard: by adding the *Geniz newsletter* plugin [6] of CMSimple. The plugin files are placed in the configuration files of CMSimple by the generator and users access to the plugin administration interface with CMSimple.

Requirement 8 This requirement (defined in Table 3.12), which imposes of offering the possibility of adding a plugin managing news related to the conference, is solved by the use of the CMSimple *Advanced news* plugin [4], as for the wizard. As all CMSimple plugins, this one is administrated with the administration interface of the CMS and its files are inserted with the configuration files of CMSimple.

Requirement 9 This requirement (defined in Table 3.13), which consists to offer the possibility to add a plugin managing forms, is solved by adding the *Advanced form* plugin [3] available with CMSimple. The configuration files are placed with other plugin files by the generator and an area in the administration interface of CMSimple is devoted to manage the installed plugins.

Requirement 10 This requirement (defined in Table 3.14) imposing to offer the possibility of adding a plugin displaying a slideshow on the homepage, is taken in account by the addition of the *PHP/SWF Slideshow* script [30]. The generator places the script in the configuration of CMSimple and places the command allowing to call the slideshow in the content file of CMSimple.

Requirement 11 This requirement (defined in Table 3.15), which consists in offering the possibility of adding up to 10 images in the slideshow, is solved thanks to the lists of attributes of each feature in Pure::variants. As we have seen for the second requirement, we can add a list of attributes for the *Slideshow* feature. These attributes of "Directory" type specify the images directory. Users can add as much attributes as they want. The generator recovers the directories and uploads the files in the appropriate CMSimple folder.

Requirement 12 This requirement (defined in Table 3.16) imposing to offer the possibility of adding 5 other templates in addition to the default template, is solved thanks to the FD. Indeed, we have added 6 features to represent the six possible templates. However, it is impossible to display a preview of the templates in Pure::variants. To solve this problem, we need to provide the preview in an external source, such as in the documentation.

Technical complications

This section presents the two technical complications that we have not succeed to overcome in the design of this solution. The first problem that we have encountered is a script problem in Pure::variants, as we have explained

in the implementation choices, and the second is a problem with the lists of attributes. We have never succeeded to execute scripts in Pure::variants, that has imposed the creation of a generator, independent of Pure::variants. The problem with the lists of attributes occurred during the generation of the FD in an *XML* file, input of the generator. Indeed, the *XML* file contains only the first attribute of the list, the others were omitted. No solution to this problem has been found for the moment.

4.2.2 Graphical user interface

In this section, we define the look and feel and the design of our solution. Here we are interested only in Pure::variants because the generator has no GUI, it is just an application executable in command lines. Pure::variants being an Eclipse plugin, we explain how it is designed.

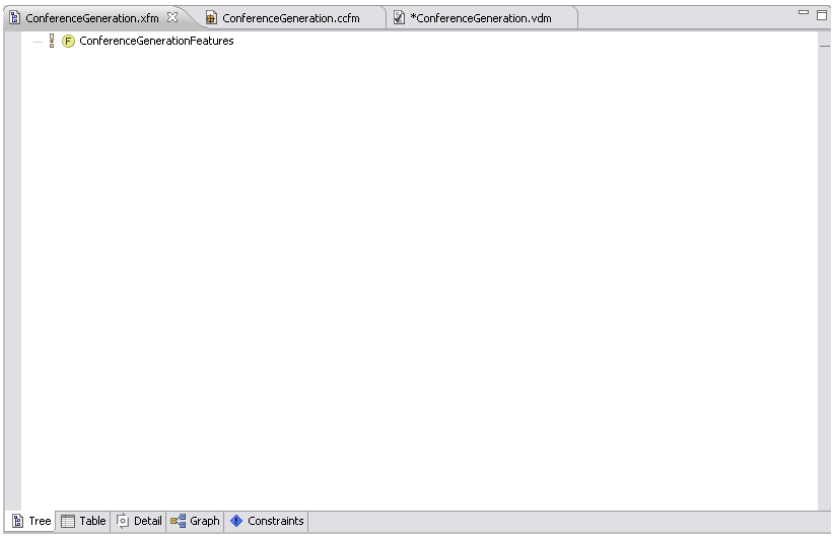


Figure 4.26: Generic approach: screenshot of a new project

When users launch Pure::variants for the first time, they have to create a standard project. When it is made, three editor windows (as shown in Figure 4.26) open allowing to edit the project: one for the FD, one for the Family Model (FM) and one for the Variant Description Model (VDM). The FD is created with a root feature. "The FM describes how the products in the product line will be assembled or generated from pre-specified components. Each component in a FM represents one or more functional elements of the

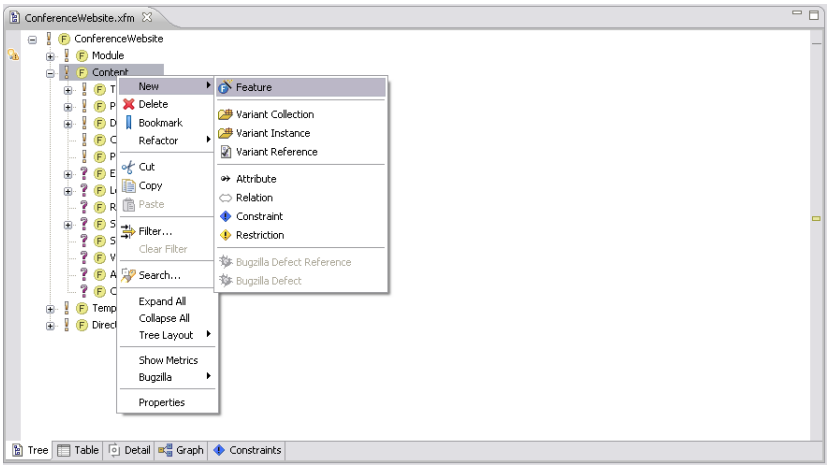


Figure 4.27: Generic approach: screenshot of a new feature addition

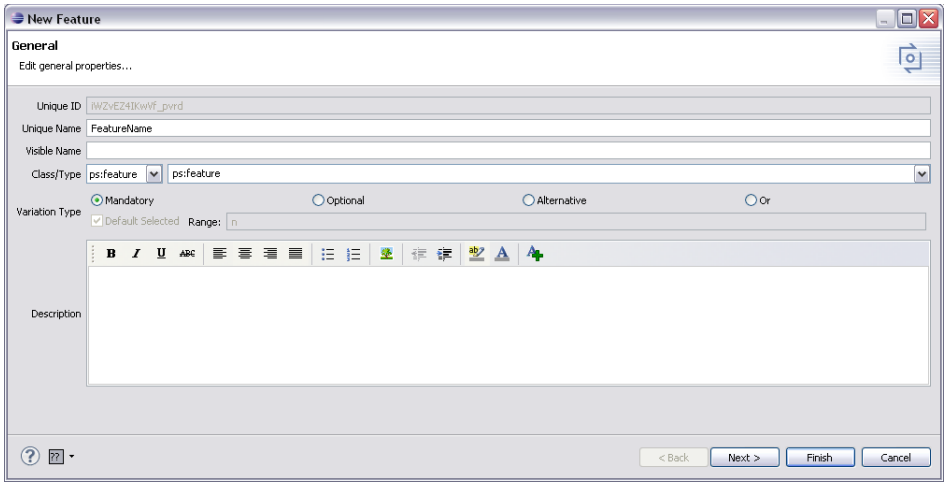


Figure 4.28: Generic approach: screenshot of details of a new feature addition

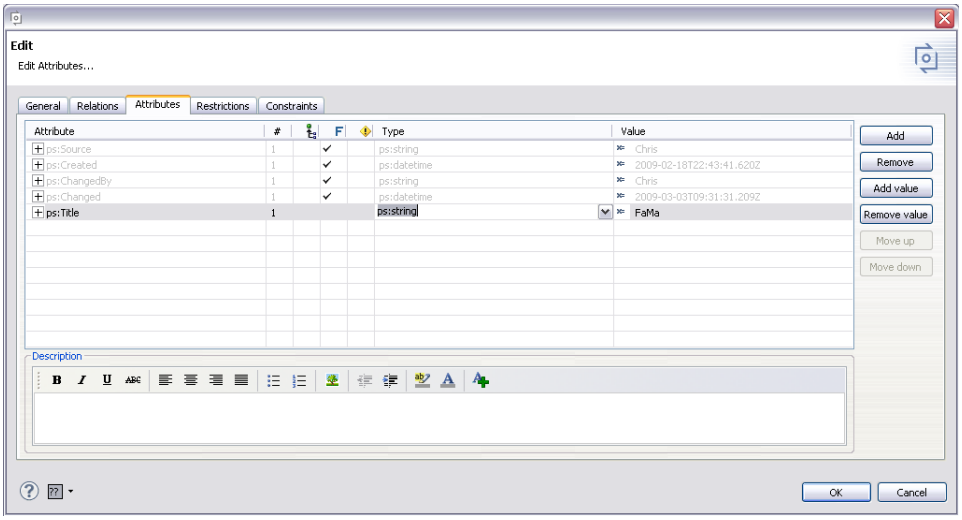


Figure 4.29: Generic approach: screenshot of new attribute addition

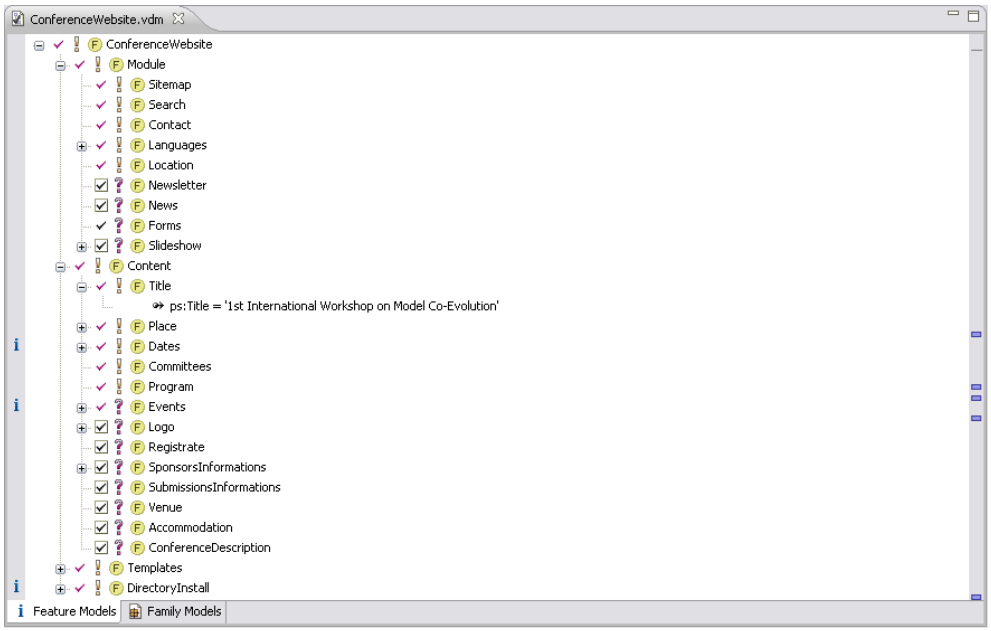


Figure 4.30: Generic approach: screenshot of features selection

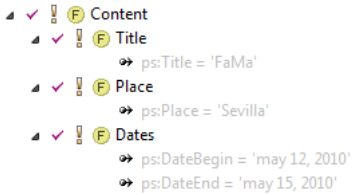


Figure 4.31: Generic approach: screenshot of attributes

Figure 4.32: Generic approach: screenshot of Pure::variants export function

products in the product line. The VDM describes the set of features of a single product in the product line " [80]. Here we especially focus on the FD and the VDM, the FD allows users to build their models and the VDM, to complete their configuration with the VDM Editor.

From the root feature, users can build their FD by adding child features to it. The addition of new features, shown in Figure 4.27 is made with a right click on the parent feature, then the selection in the menu of *New* and *Feature*. At that time, a new window (visible in Figure 4.28) opens to provide the feature characteristics as the *unique name*, possibly the *visible name*, the *variation type* (*mandatory*, *optional*, *alternative* and *or*) and a *description*. The *alternative* and *or* variation type allow to set the cardinalities. Once the feature is created, attributes can be assigned to it. A right click, the selection of *New* and *Attributes* opens a window (visible in Figure 4.29) to add new attributes to the chosen feature, and it is also possible to edit the *general* information and the *relations*, create *restrictions* or *constraints*. To add attributes, users have to click on the *Add* button and edit the required information. Information such as the *attribute* name, the *type* and the *value* must be provided. Users can specify a list of values.

In repeating the same operation for each feature, the CWPL can be built and users can use the VDM Editor to select the desired features, as shown in Figure 4.30. However, in our case, we have already defined the CWPL FD which can be reused. For that, user can import it and use it in the VDM Editor. A mandatory feature is represented with a red cross which avoids its deselection whereas an optional feature can be selected or deselected. Attributes and their values are also displayed on the model, as shown in Figures 4.30 and 4.31.

When the configuration is finished, users provide the configured FD to the generator with the function of exportation. For that, they have to make a right click on the project or the VDM file and select *Export*. They have to select to export in the *XML* format (as shown in Figure 4.32) and then specify the destination folder which must be those of the generator. Once it is completed, users can launch the generator on command lines and the generation begins.

4.3 Comparison

This section is dedicated to the comparison of the two approaches developed above in order to highlight their advantages and their drawbacks. For that we use the *ISO 9126* standard. *ISO 9126* is an international standard for the evaluation of software quality [84]. This standard has the fundamental objective to address some of the well known human biases which can have an

effect on the perception and the delivery of a software development project. Priorities that have not any clear definitions of success or that change after the begin of a project are included in these biases. *ISO 9126* tries to develop a common understanding of the project's objectives and goals. It categorizes software product attributes into six characteristics, which are further subdivided into sub-characteristics [68].

First, we define the comparison criteria of *ISO 9126*. Then we analyse the solutions depending on the defined criteria.

4.3.1 Comparison criteria

The software product attributes of the *ISO 9126* standard are categorized into the six following characteristics: *Functionality*, *Reliability*, *Usability*, *Efficiency*, *Maintainability*, and *Portability*. In this standard, criteria are named characteristics and each of them has further sub-characteristics.

We confront the approaches to the 21 characteristics (summarized in Table 4.2) of the *ISO 9126* standard, which are defined below.

The *Functionality* characteristic is subdivided into 5 sub-characteristics: *Suitability*, *Accuracy*, *Interoperability*, *Compliance*, *Security*. In *ISO 9126*, the *Suitability* is the essential functionality characteristic that is the capability of the software product to provide appropriate functions for specified tasks [89]. The *Accuracy* is the capability of the software product to provide correct results with the needed degree of precision [21]. The *Interoperability* is the ability of the software product to interact with other specified components or systems. The *Compliance* is the capability of the software product to adhere to standards, conventions or regulations in laws. The *Security* is the ability of the software product to prevent unauthorized access (accidental or deliberate) to the software functions.

The sub-characteristics such as *Maturity*, *Fault tolerance* and *Recoverability* are parts of the *Reliability* characteristic. The *Maturity* is the ability of the software product to avoid failure. The *Fault tolerance* is the capability of the software product to withstand in cases of software defects. The *Recoverability* is the ability of the software product to bring back a failed system and recover the data affected by the failure.

The *Usability* characteristic is composed of the *Understandability*, *Learnability*, *Operability* sub-characteristics. The *Understandability* is the capability of the software product to enable the user to understand how it can be used for particular tasks. The *Learnability* is the ability of the software product to enable the user to learn its application. The *Operability* is the capability of the software product to be easily operated and controlled by a user.

The *Efficiency* characteristic is decomposed into two others: efficiency in relation to *Time behaviour* and to *Resource behaviour*. The efficiency is the capability of the software product to provide appropriate performance, in terms of response times, turnaround, network, etc. for the *Time behaviour*, and in terms of resources used such as memory, CPU, disk and network usage for the *Resource behaviour*.

The *Maintainability* is subdivided into *Analysability*, *Changeability*, *Stability* and *Testability* sub-characteristics. The *Analysability* is the capability of the software product to identify the deficiencies or causes of failures in the software. The *Changeability* is the ability of the software product to facilitate the implementation of modifications. The *Stability* is the capability of the software product to avoid negative effects caused by modifications in the software. The *Testability* is the ability of the software product to enable software changes to be tested.

The *Portability* characteristic has four sub-characteristics such as *Adaptability*, *Installability*, *Conformance* and *Replaceability*. The *Adaptability* is the ability of the software product to be adapted to new specifications or operating environments. The *Installability* is the capability of the software product to be installed in a specified environment. The *Conformance* is the capability of the software product to be in correspondence with current requirements of legislation, specified standards, or terms of a contract. The *Replaceability* is the ability of the software product to be exchanged with another specified software product in the same environment.

However, we consider that, on the one hand, the *ISO 9126* criteria are not sufficient to compare some aspects. Indeed, some important criteria for us such as costs, help, configuration checking, input format, etc. are not taken in account. And in the other hand, they evaluate criteria that we consider not relevant for our case such as the security, reliability, etc. We can so refine our comparison criteria in adding new ones and in deleting others.

We have chosen to add five new criteria which are the following: *cost*, *input format*, *configuration check*, *possibility to add specific features*, *help*, and *feedback*. The *cost* is the required cost to purchase the application. The *input format* is input type required in order to use the tool. The *configuration check* is the availability in the application of a check of the user's configuration validity. The *help* criterion is the availability of a help to use the application. The last is the *feedback on errors* which is the availability of a system of feedback to warn the user when an error occurs.

These criteria have been chosen because we consider that they are important to judge of the use quality of the applications. We are interested to costs that are needed to use them, we favour free solutions. As users provide the

Table 4.2: Summary of the ISO 9126 standard criteria

Functionality	Suitability
	Accuracy
	Interoperability
	Compliance
	Security
Reliability	Maturity
	Fault tolerance
	Recoverability
Usability	Understandability
	Learnability
	Operability
Efficiency	Time behaviour
	Resource behaviour
Maintainability	Analysability
	Changeability
	Stability
	Testability
Portability	Adaptability
	Installability
	Conformance
	Replaceability

input of the applications, they must understand it to start with the system. So we can evaluate the ease to take in hand the applications. When the website is generated, we cannot envisage errors in the configuration, so the applications check the configurations made by users. Finally, we have an interest for the help and the feedback provided by the applications. The help is useful to start with the system or for any question on its working and the feedback allows to guide the user in case of error.

We have deleted eleven criteria of the standard because we consider that they are not relevant in our case. The two first are *functionality* sub-characteristics: *compliance* and *security*. As the conference website generation does not need to adhere to standards, conventions, etc., it is not useful to keep the *compliance* characteristic. It is the same for the *security* because the applications do not contain confidential information and so they do not prevent unauthorized access. We have also deleted the *reliability* characteristic and its three sub-characteristics: *maturity*, *fault tolerance* and *recoverability*. The

two solutions have been developed within the framework of this master thesis so they have no *maturity*. They have not been put in application in real conditions so it is impossible to evaluate their *reliability* and *fault tolerance*. At the end of the development, we have tested the functionalities and no failures occurred. The *efficiency* criterion is also removed because it is not relevant for our case as the generation is completed in a correct time without monopolize all the resources of a computer. The application specific and generic solutions can run on any operating system with a minimum memory space and CPU usage and the generation the conference website takes only some seconds but depends on the number of features selected. From the point of view of the *maintainability*, we have removed the *stability* and *testability* characteristics because they are also not useful. It is not possible to compare both solutions on *stability* and *testability* because no system has been developed for these tasks. Indeed, they are not able to avoid negative effects caused by modifications and the test must be made directly by testing different configurations. The last two elements deleted are the *conformance* and the *replaceability* because respectively there are actually no requirements of legislation, specified standards, etc. in the area of conference website, and there are not interest to replace the application specific and generic solutions. As they are new solutions, for a new task executed by no tools on the market, it is not possible to replace them. Table 4.3 summarizes the different criteria on which we base our comparison.

4.3.2 Analysis

In this section, we outline the major differences of both approaches developed in the two first sections of this chapter on basis of the criteria defined before. For each criterion, we give an appreciation: *good* if the criterion is completely met by the application, *average* if it is partially met and *bad* if it is not good met.

The first criterion of the comparison (summarized in Table 4.4) confronting the solutions is the *suitability* characteristic. As the application specific solution has been specially designed for the website generation, its *suitability* is good whereas it is average for the generic solution because it cannot provide all the generation process. Indeed, Pure::variants cannot execute itself the generation (see the technical complications in Section 4.2.1). The *accuracy* of both approaches is good because they provide a correct result, the conference website possesses the selected features and is usable. At the level of the *interoperability*, it is good for both approaches because they are able to interact with other specified components or systems as they are written in Java.

Table 4.3: Summary of the refined ISO 9126 standard criteria

ISO 9126	Functionality	Suitability
		Accuracy
		Interoperability
	Usability	Understandability
		Learnability
		Operability
	Maintainability	Analysability
		Changeability
	Portability	Adaptability
		Installability
Additional	Cost	
	Input format	
	Configuration check	
	Help	
	Feedback on errors	

At the level of the *usability*, *understandability* and *learnability* of the application specific approach are good because it has a simple and intuitive GUI guiding users through the website generation process. The generic approach has average capabilities in terms of *understandability* and *learnability* because of Pure::variants which is relatively complex to use and thus to learn and the generator have to be used in command lines. The application specific has a good *operability* because it can easily be operated and controlled by users. Users just have to launch the application and follow the interfaces. The generic solution is based on an *Eclipse* plugin (Pure::variants) and a generator, so two separate tools to use. Its operability is thus average.

For the *maintainability* characteristic, we confront the two solutions to two criteria. The first one is the *analysability* which is bad for the both solutions because nothing has been implemented to identify their deficiencies or causes of failures. The second one, the *changeability* is good for the application specific approach because its source code is relatively simple, allowing to easily implement modifications. The generic approach has an average *changeability* because of Pure::variants. Indeed, Pure::variants is a plugin integrated in Eclipse with a complex source code which makes difficult any modification. For example to personalize the look and feel of the application by displaying images to make the choice of features, it implies complex modifications in Pure::variants whereas it is easier for the wizard.

The next characteristic is the *portability* which is composed of the following sub-characteristics: *adaptability*, *installability*. The *adaptability* can be qualified of good for the application specific solution because its source code and GUI can easily be adapted to new specifications or operating environments as they are written in Java. For instance, we can easily display images to present the templates to choose for the website. The *adaptability* is average for the application specific solution because in case of addition or modification of features, its GUI and the generation algorithm must be adapted. The *adaptability* of the generic approach is good because in Pure::variants we can easily adding, modifying or deleting features, we have just to update the generation algorithm. One of the constraints on the solution made in the requirement engineering analysis (see Section 3.3.4) is to run on any operating system and thus be easily installed. However, the application specific approach consists in one tool whereas the generic contains two tools (Pure::variants and generator). The *installability* of the first one is good but average for the second one.

On the point of view of the *cost*, there is no real difference between the approaches. Indeed, the application specific and generic solutions are free. The generic solution uses Pure::variants which is free only for non-commercial use, which is our case.

We are now interested with the *input format*. It is completely different for both solutions. For the wizard, it is simple comboboxes to select in functions of choices made by users whereas the input of Pure::variants is a FD and an *XML* file for the generator.

At the level of the *configuration check*, the both approaches complete it, but differently. The validity of the application specific approach is provided by its GUI which avoids any error and by FaMa-FW. The generic approach verifies at each moment the configuration and corrects it automatically.

At the point of view of the *help*, the wizard offers a support to help users to select the features by displaying tips on each feature when they dispose the mouse on it. The other application offers also help thanks to Pure::variants in which it is possible to give a description for each feature.

Finally, no errors are possible with both tools. The first one gives a *feedback* on invalid dates (for example, begin date before end date) or not selected mandatory features, when users go from an interface to another. The second one makes the correction automatically without warn users such as for the features required by others, they are automatically selected. For other cases, it provides also *feedback* when errors occurs.

We have outlined the major differences of the application specific and generic approaches, we will discuss them extensively in the next chapter (see

Table 4.4: Summary of the refined ISO 9126 standard comparison

ISO 9126	Functionality	Suitability	Good	Average
		Accuracy	Good	Good
		Interoperability	Good	Good
	Usability	Understandability	Good	Average
		Learnability	Good	Average
		Operability	Good	Average
	Maintainability	Analysability	Bad	Bad
		Changeability	Good	Average
	Portability	Adaptability	Average	Good
		Installability	Good	Average
Additional	Cost		Free	Free for non-commercial use
	Input format		Comboboxes to select	Feature diagram
	Configuration check		Yes	Yes
	Help		Yes	Yes
	Feedback on errors		Yes	Yes

Chapter 5) in order to develop a unified approach which combines the advantages of both.

Chapter 5

Towards a "unified" approach

This chapter presents the ideal solution combining the advantages of the application specific and generic approaches presented in the previous chapter. First, we begin to detail deeper these two approaches. Second, we fix the characteristics which require more attention and we prioritize them. Third, on basis of the characteristics highlighted previously, we outline a comprehensive description of what a unified approach should offer and finally we discuss its limitations.

5.1 Discussion of previous approaches

In this section, we start from the comparison made in the previous chapter (see Section 4.3) to highlight the characteristics of both approaches which are satisfactory in our context.

The first criterion of comparison (summarized in Table 4.4) is the *suitability* characteristic. We cannot envisage a solution that is not able to entirely provide the appropriate functions for website generation. The good *suitability* of the application specific approach fully satisfies this property whereas it is not satisfied by the feature-based solution. Indeed, it is not possible to integrate the generation algorithm in Pure::variants imposing to solve this problem with a generator which executes the website generation.

The second criterion is the *accuracy*. It is an important criterion because the solutions must provide a website ready to use and correctly configured. Both solutions have a good *accuracy* allowing to satisfy this criterion.

The third criterion is the *interoperability* which is good for both approaches. As they may interact with conference management systems such as EasyChair (see constraints defined in Section 3.3.4), it is useful to respect this constraint

which is satisfied by both solutions.

The two following criteria are the *understandability* and the *learnability* of the application specific and generic approaches. These criteria allow to evaluate the user-friendliness of both solutions, a characteristic for which we are sensitive. The website configuration must be as much easy as possible, to easily be used by the maximum of users. The *understandability* and the *learnability* of the application specific approach is good, which is satisfactory for our context. However, the generic solution with its average user-friendliness (average level for *understandability* and *learnability*) is not entirely satisfying because it uses Pure::variants which is a relatively complex and not completely intuitive tool.

The sixth criterion is the *operability* of both approaches. This criterion must be fully satisfied because we are looking for a solution which can easily be operated and controlled by users, and requiring the minimum knowledge. As it is a stand-alone application, the application specific approach satisfies this criterion because it has a good *operability*. On the other hand, the average *operability* of the generic approach is due to the use of two tools (Pure::variants and generator) to reach our objective. This solution is not practical.

The next criterion is the *analysability* which is bad for both approaches because nothing has been implemented to identify their deficiencies or causes of failures. These bad levels are satisfying in our case because we do not need that the solutions are able to identify their failures. Indeed, they must be validated after their development in order to be without any bug.

The eighth criterion is the *changeability*. The application specific solution has a good *changeability* allowing to satisfy this criterion because its implementation can be quickly changed. Indeed, the core of this solution consists in a generation algorithm composed of modules. These modules can easily be added, modified or deleted. However, the generic approach has an average level of *changeability* because Pure::variants is a plugin integrated in Eclipse with a complex source code making difficult any change in its functionalities.

The ninth criterion is the *adaptability* which is respectively average and good for the application specific and generic approaches. We are looking for a solution which can be quickly adapted because the features of the conference website can be modified or deleted, or new ones added. The first approach does not satisfy this criterion whereas the second one satisfies it. Indeed, for both of them, the generation algorithm must be updated but in addition, the GUI of the application specific solution must also be adapted. The GUI of the generic approach is managed by Pure::variants allowing to display any change in the conference website FD.

The tenth criterion is the *installability*. The application specific approach

satisfies this criterion because it consists in an executable application which is able to run on different operating systems. This last condition is imposed by the constraints made in the requirement engineering analysis (see Section 3.3.4). The generic approach do not satisfies this criterion because the deployment of this solution is more complicated. Indeed, the install of Pure::variants requires beforehand the install of Eclipse. In addition, users have to be accustomed to the install of plugin in Eclipse, otherwise they have to consult tutorials on how to proceed.

The eleventh criterion is the required *cost* to purchase the applications. As we are looking for a free solution to automate the creation on conference website, application specific and generic approaches satisfy this criterion because both are free.

The twelfth criterion is the *input format* which allows to highlight the required input to use the application. We desire that the conference website generator requires the minimum knowledge to use it and so to provide the input. In this context, the application specific approach satisfies this criterion because no knowledge is required, users have just to select the desired features. On the other hand, the input of the generic approach is a FD which imposes to know how to interpret it. This not satisfies the criterion.

The following criterion is the *configuration check*. In the configuration of conference websites, some errors may be possible such as selecting two times the same languages, or selecting the registration page without selecting the form plugin, etc. So it is important to check that there are no errors in the configuration. For that, a *configuration check* must be applied during the configuration process or once it is finished. Both solutions satisfy this criterion because they provide a such functionality.

The fourteenth criterion is the *help* provided to users in the configuration activity. Each feature of the conference website may be described in order to help users to know what they select. The application specific and generic approaches provide the possibility to display the features descriptions, which is satisfactory.

The last criterion is the ability of the approaches to provide *feedbacks* when errors occur. As we have seen with the *configuration check*, in the configuration of conference websites, some errors may occur. So it is important that users receive *feedbacks* on their errors, allowing them to correct their configuration. As both solutions provide *feedbacks*, they satisfy the criterion.

Table 5.1 sums up for each approach (application specific and generic) which criteria it satisfies. When the approach satisfies the specified criterion, a *Yes* is attributed, *No* otherwise.

Table 5.1: Summary of the satisfaction of the application specific and generic approaches in relation to the specified criteria

Type	Characteristic	Sub-characteristic	Specific	Generic
ISO 9126	Functionality	Suitability	Yes	No
		Accuracy	Yes	Yes
		Interoperability	Yes	Yes
	Usability	Understandability	Yes	No
		Learnability	Yes	No
		Operability	Yes	No
	Maintainability	Analysability	Yes	Yes
		Changeability	Yes	No
	Portability	Adaptability	No	Yes
		Installability	Yes	No
Additional	Cost		Yes	Yes
	Input format		Yes	No
	Configuration check		Yes	Yes
	Help		Yes	Yes
	Feedback on errors		Yes	Yes

5.2 Change priorities

Here, we identify the characteristics of the previous approaches that we have to change for the solution proposal as well as their priorities. We have chosen to specify two levels of priority for each criterion: urgent or secondary. An urgent priority for a criterion means that the solutions proposed by the previous approaches must be changed urgently by a new solution. A secondary priority means that in the previous approaches, a well designed solution can be taken back.

In the previous section, we have seen that together, the application specific and generic approaches satisfy all the characteristics (as shown in Table 5.1). However, for some criteria, one of both approaches do not satisfy, so we attribute an urgent priority to these criteria. As shown in Table 5.2, eight characteristics have an urgent priority because we have to make a choice between the solutions proposed by both approaches. These characteristics are the following: *suitability*, *understandability*, *learnability*, *operability*, *changeability*, *adaptability*, *installability*, and finally *input format*. The fact that a criterion has a secondary priority means that application specific and generic

approaches propose more or less the same solution, and so we can reuse it.

The first criterion having an urgent priority is the *suitability*. The solution must provide all the appropriate functions for the generation of conference websites, and only that ones. The second and third criteria are the *understandability* and *learnability*. They represent the user-friendliness of the application, which means that more an application is user-friendly, more it is easy and rapidly usable. Users can thus complete conference websites generation in a minimum amount of time. On the other hand, the solution must have an interface allowing to quickly learn and use all the functions of configuration and generation. It must avoid other functions which are not in relation with conference websites generation. For the fourth criterion, the *operability*, the solution must easily be operated and controlled, in requiring a minimum knowledge. For that, it must avoid to use different tools to generate the conference website. The fifth criterion is the *changeability*, and imposes that the implementation of the solution must be easily extended. Indeed, the solution must allow to evolve by the addition of new functionalities, or by the modification or deletion of others. This criterion influences the solution design, so it must be taken in account early in the development stage. The sixth criterion which must be urgently changed is the *adaptability*. Indeed, the features of conference websites may change (addition, modification, deletion of features), so the solution must be easily and quickly adapted to these modifications. For the *installability* criterion, the seventh criterion, the solution must be installed on any operating system and in requiring the minimum of handling. We must avoid the installations requiring to install other tools to be ready to use. The eighth and last criterion is the *input format* for which we must avoid formats that require knowledge to be understanded.

5.3 Solution proposal

In this section we make a solution proposal for a unified approach using the criteria that we have prioritized in the previous section. As we have seen, we can distinguish two types of criteria: those that must be urgently changed and those that are secondary. For each criterion, we detail what to take and what to delete of the both previous approaches and possibly what to add to obtain the unified approach. First, we describe the criteria having urgent priorities and then the others.

The *suitability* is the first urgent change to capture in the solution proposal. We keep the solution developed in the application specific approach in designing an application especially dedicated to the conference website generation. This allows to avoid the problem seen with Pure::variants in the generic

Table 5.2: Summary of the priorities assigned to characteristics

Type	**Characteristic**	**Sub-characteristic**	**Priority**
ISO 9126	Functionality	Suitability	Urgent
		Accuracy	Secondary
		Interoperability	Secondary
	Usability	Understandability	Urgent
		Learnability	Urgent
		Operability	Urgent
	Maintainability	Analysability	Secondary
		Changeability	Urgent
	Portability	Adaptability	Urgent
		Installability	Urgent
Additional	Costs		Secondary
	Input format		Urgent
	Configuration check		Secondary
	Help		Secondary
	Feedback on errors		Secondary

approach which contains other functions that we do not need to accomplish the specified task. These other functions may confuse users who may be lost in the functionalities of the solution. We need of five functions: opening of feature diagrams, selection of features, deselection of features, generation of conference website, and providing help.

On the point of view of the *understandability* and *learnability*, we take the solution proposed by the application specific approach which consists in a wizard with an intuitive interface. The wizard can guide users in the conference websites configuration throughout different dialog boxes. We avoid the generic approach which has an interface too complex and several functions not relevant for the generation of websites.

We are then interested in the *operability* criterion. We keep also the application specific approach which consists in a stand-alone application and we avoid the solution of the generic approach. The solution proposal must be an integrated application which contains all the required functionalities for conference websites generation. Indeed, the use of different tools in the same solution complicates the ease of operating it.

For the *changeability* criterion, we keep the solution proposed by the application specific approach which consists, as we have already seen, in a stand-alone application especially designed for the conference websites generation. It

has a source code which is not too complex and well documented for allowing to easily make changes. We avoid the solution of the generic approach which is an Eclipse plugin complicating any change in the source code because the integration in Eclipse has to be managed in addition .

On the point of view of the *adaptability*, we take in the generic approach its independence in relation to the GUI. So we propose that the solution proposal automatically generates a wizard in which all the features of the FD are displayed. Users keep thus an intuitive and simple interface automatically managed by the tool. They have just to add or modify the desired features and adapt the generation algorithm in function of the modifications but deal not with the associated GUI components. We have to avoid the GUIs which are implemented manually such as for the application specific approach.

For the *installability* criterion, we keep the solution developed in the application specific approach. This solution consists in an executable stand-alone application. So it can be executed on any operating system. We avoid the generic approach which consists in a Eclipse plugin requiring the install of Eclipse in addition to the generator.

The last urgent criterion is the *input format*. We keep the solution of the application specific approach which consists in displaying the features on the interface and allowing their selection. So we abandon the solution proposed by the generic approach consisting to show the features under the form of a FD. This solution confirms the proposition already made, to implement a solution proposal that generates automatically the interface (wizard) displaying the different features. This solution allows to avoid additionally any knowledge to carry out the configuration.

The first secondary criterion is the *accuracy*. For that, we keep the solution developed in both approaches which consists in a well implemented and tested generation algorithm. This algorithm allows to generate conference websites ready to use, correctly configured and without errors.

On the point of view of the *interoperability* criterion, we keep both previous approaches by developing the unified solution using the Java language. Indeed, Java can be easily operated with other applications. In addition, this criterion is a constraint (defined in Section 3.3.4), that must be satisfied.

For the *analysability* criterion, we estimate that it is not useful in the development of this unified approach. Indeed, the implementation of a functionality which identifies deficiencies is not pertinent in our case because our solution proposal will be exempt from deficiencies and bugs.

On the point of view of the *cost* criterion, we cannot envisage the development of a paying solution for our unified approach. So we propose a solution entirely free such as the application specific and generic approaches.

At the level of the *configuration check*, as we have chosen to develop an application which generates a wizard, we propose the same solution to check the configuration than the application specific approach. The interface must be correctly generated by the unified solution avoiding any error of configuration. All the checks for mandatory information, required features, etc. must be carried out dynamically during the configuration process.

On the point of view of the two last criteria, *help* and *feedbacks*, both approaches, application specific and generic, were equals for helping users in their choices and warning them when an error of configuration is made. So, we keep this solution which consists to provide a description of each feature and provide feedback when configuration errors are made. However, we propose to add the obligation to specify the description of each feature added.

To summarize the main characteristics of our solution proposal, it is a stand-alone application executable on any operating system. It allows to generate automatically a wizard which allows users to configure their conference website in an intuitive interface and require no knowledge to use it. This interface avoids any errors (the configuration is validated at each moment) and provides help and feedback to help users in their selection or to complete some information. In addition, all the features are correctly defined. If the CWPL evolves, the wizard is updated automatically. Finally, it can easily be adapted and interoperated with other applications as it is written in the Java programming language.

5.4 Known limitations

In this section, we discuss the limitations of the unified approach proposed in the previous section.

The first limitation we can highlight is situated at the level of the *changeability*. Indeed, our unified approach proposes to have source code well documented and not too complex in order to facilitate the implementation of modifications, but if we implement all the proposed functionalities, the source code become complicated and the *changeability* can decrease.

The second limitation is situated at the level of the *configuration check*. As the solution proposal consists in an application which automatically generates a wizard to guide users during the configuration process, its interface must automatically check the validity of the configuration. But for that, users have to specify, during the addition of a new feature, what checks operate and on which objects. It implies to code these checks in the application implementation language (here Java) which requires knowledge and complicates the use

of the solution. So the application lost in quality in terms of *operability* and *understandability*.

The third limitation is linked to the *understandability* and *learnability*. As the unified solution generates automatically a wizard in function of the FD, it manages also the display of the different features on its interface. But we are not sure that this display will be coherent. For instance, some features require to be grouped, or others ask a textual entry instead of to be selected, etc. This problem may decrease the coherence and user-friendliness and thus the *understandability* and *learnability* of the solution.

The fourth limitation is that this approach has not been implemented yet by lack of time. All the solutions to criteria discussed for this solution proposal are purely theoretical. As we have not implemented it, we have not tested it, so we are not able to check if our propositions can match all the criteria in practice.

Chapter 6

Conclusions

We conclude this work by first synthesizing the objectives and research questions we tried to tackle. Then, we review our contributions and outline the limitations of our approach. Finally, we discuss open perspectives and show how our work could adjust to a variety of application domains.

6.1 Summary

Scientific conferences allow active researchers and engineers to state and communicate their works. The organization of conferences passes in particular by the creation of a website presenting their practical modalities. As conferences have a well defined organization with recurring operating modes, we can envisage to represent them using software product lines. The objective of this thesis was to provide a solution to automate the creation of conference websites using software product lines approach.

We have so presented two different approaches to automate the creation of conference websites. The first one, an application specific approach consisted in the development of a wizard allowing to complete the configuration with an intuitive interface. The wizard provides a succession of screens presenting website characteristics that users have to select. The second was a generic approach which consisted to use a variability modelling and configuration tool to make the configuration and then use this configuration to generate the website.

After having analysed those both approaches, we have confronted them in relation to different criteria. This comparison allowed us to highlight their strengths and weaknesses and the need to develop a unified approach combining their advantages.

6.2 Contributions

Our master thesis focuses on the automation of the creation of products using the software product lines approach. We have noticed that no current proposals accomplished the automation of the creation of product. So, we concluded that there is a gap that has to be filled in. We have chosen to illustrate this problem in the conference management field using conference websites as products. We have used feature diagrams to represent the conference website product line and all its characteristics. The conference website product line was the basis of the development of three solutions.

The first solution is a wizard in which all the features of the conference website product line are described. This solution provides an intuitive representation of the conference website product line to be comprehensible for users. They have just to select in the succession of interfaces the desired features.

The second solution tackles the problem using another approach. It uses a variability modelling and configuration tool allowing to select and configure the required characteristics of a website within the conference website product line. In the tool, the conference website product line is represented by a feature diagram that users use for the selection of the desired features.

The third approach combines the strengths of both first solutions. It consists in a solution which dynamically generates a wizard containing all the features of the conference website product line in function of the feature diagram. If the product line evolves, the wizard is automatically updated.

6.3 Limitations

Following, we discuss of the limitations of the main decisions that we have taken in this master thesis.

The first decision was to illustrate the problem of automation of the creation of the core assets available in the software product line using conference websites. Indeed, our solutions have been only validated in the conference management field whereas we could use other fields.

The second decision was to use a feature diagram to represent the conference website product line. Indeed, feature diagrams are a widespread means to represent software product lines. The problem with the feature diagram approach is that they have not actually been tested by non-experts. Other variability modelling languages than feature diagrams can be used to represent software product lines and the variability between products.

The third decision was to use a content management system for website generation. This solution seemed to be useful because a content management

system has a ready to use structure and is easily updatable and configurable. With a content management system, we have avoided the development of a complete website in *PHP*, *HTML*, etc. The configuration was also simplified thanks to the content management system structure because only two files had to be modified. However, we could use other solutions than a content management system.

The fourth decision was to propose a unified approach having the advantages of the application specific and generic approaches developed before. Lacking of time to implement this approach, we have made a theoretical solution proposal which has thus not been tested. So we are not able to prove that our solution can be efficient in practice and respect all the criteria we have defined.

6.4 Perspectives

Here we discuss of the different perspectives of our work that we have identified.

As we were short in time, we had not enough time to implement the unified approach (described in Chapter 5). A perspective in the short term will be to implement it in respecting the priorities assigned to each characteristic.

A second perspective will be, once the unified approach is implemented, to validate the three approaches in order to identify which one performs best according to the criteria that we have defined in Section 4.3.1.

A third perspective will be to apply these solutions to other domains than conferences. We can think to domains such as mass production of cars, etc. Indeed, users might select the desired features of a car and the application directly commands the production line to produce the car with the specified characteristics.

Bibliography

[1] Cms definition. http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci508916,00.html.

[2] Cms matrix website. http://www.cmsmatrix.org.

[3] Cmsimple advanced form plugin. http://www.cmsimplewiki.com/doku.php/plugins/advancedform.

[4] Cmsimple advanced news plugin. http://www.cmsimplewiki.com/doku.php/plugins/advancednews.

[5] Cmsimple forum website. http://www.cmsimple.com/forum.

[6] Cmsimple geniz newsletter plugin. http://www.cmsimplewiki.com/doku.php/plugins/geniznewsletter.

[7] Cmsimple mailing list. http://freshmeat.net/cmsimple.

[8] Cmsimple official website. http://www.cmsimple.org.

[9] Cmsimple website. http://www.cmsimple.com.

[10] Cmsimple wiki. http://www.cmsimplewiki.com.

[11] Colibri conference management system. http://www.nongnu.org/colibri/index.html.

[12] Comparison of cms. http://en.wikipedia.org/wiki/Comparison_of_content_management_systems.

[13] Easychair conference system. http://www.easychair.org/.

[14] Fama fw website. http://www.isa.us.es/fama/.

[15] Fmp website. http://gsd.uwaterloo.ca/projects/fmp-plugin/.

[16] Ibm rational software architect website. http://www-01.ibm.com/software/awdtools/architect/swarchitect/.

[17] Ibm rational software modeller website. http://www-01.ibm.com/software/awdtools/modeler/swmodeler/.

[18] Ieee recommended practice for software requirements specifications. http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=15571.

[19] Info m431 course. http://www.fundp.ac.be/etudes/cours/page_view/INFOM431/.

[20] Internet calendaring and scheduling core object specification (icalendar). http://tools.ietf.org/html/rfc2445.

[21] Iso 9126 software quality characteristics. http://www.sqa.net/iso9126.html.

[22] Javadoc swing library. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html.

[23] Jcalendar website. http://www.toedter.com/en/jcalendar/index.html.

[24] Klariti: Software requirements specification template. http://www.klariti.com/Software-Requirements-Specification-Template/.

[25] Opencms on cmswatch. http://www.cmswatch.com/Feature/152-OpenCms-6.

[26] Opencms website. http://www.opencms.org.

[27] Opencms wiki. http://www.opencms-wiki.org.

[28] Openconf. http://www.openconf.com/.

[29] Php-nuke official website. http://phpnuke.org.

[30] Php/swf slideshow website. http://www.maani.us/slideshow/.

[31] Pure::variants website. http://www.pure-systems.com/pure_variants.49.0.html.

[32] Requiline website. http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline/.

[33] Spip blog. http://www.spip-blog.net.

[34] Spip community website. http://www.spip-contrib.net.

[35] Spip forum website. http://forum.spip.org.

[36] Spip mailing lists. spip-core@rezo.net, spip-en@rezo.net.

[37] Spip website. http://www.spip.net.

[38] Srs definition. http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci1243658,00.html.

[39] Tech terms computer dictionary. http://www.techterms.com/definition/wizard.

[40] Tinymce website. http://tinymce.moxiecode.com/.

[41] Uml website. http://www.uml.org.

[42] Uml's sequence diagram. http://www.ibm.com/developerworks/rational/library/3101.html.

[43] Volere requirements specification template. http://www.volere.co.uk/template.htm.

[44] W3c website. http://www.w3.org/.

[45] Webgui forums website. http://www.webgui.org/forums.

[46] Webgui videos library. http://www.webgui.org/webgui/tv.

[47] Webgui website. http://www.webgui.org.

[48] Webgui wiki. http://www.webgui.org/community-wiki.

[49] Wizard definition. http://vdict.com/wizard,6,0,0.html.

[50] Xpath website. http://www.w3.org/TR/xpath.

[51] T. Asikainen, T. Männistö, and T. Soininen. Kumbang: A domain ontology for modelling variability in software product families. In *Advanced engineering informatics journal*, 2007.

[52] Timo Asikainen. Modelling methods for managing variability of configurable software product families, 2004.

[53] David Benavides. *On the automated analysis of Software Product Lines using Feature Models: A framework for developing automated tool support.* PhD thesis, University of Seville, 2007.

[54] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. Fama: Tooling a framework for the automated analysis of feature models. Technical report, University of Seville, 2007.

[55] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. Technical report, University of Seville, 2005.

[56] Danilo Beuche and Mark Dalgarno. Software product line engineering with feature models.

[57] J. Bosch. In proceedings of the 2nd groningen workshop on software variability management: Software product families and populations. In *Workshop on Software Variability Management*, 2004.

[58] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, Henk Obbink, and Klaus Pohl. Variability issues in software product lines. In *Software Product-Family Engineering*, 2002.

[59] CEDITI. A kaos tutorial. Technical report, CEDITI, 2003.

[60] CEDITI. Objectiver: résumé des notations. Technical report, CEDITI, 2005.

[61] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[62] K. Czarnecki, M. Antkiewicz, Chang Hwan Peter Kim, Sean Lau, and Krzysztof Pietroszek. fmp and fmp2rsm: Eclipse plug-ins for modeling features. using model templates. Technical report, University of Waterloo, 2005.

[63] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. Technical report, University of Waterloo, 2004.

[64] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. Technical report, University of Waterloo, Canada and University of Applied Sciences Kaiserslautern, Zweibrücken, Germany, 2005.

[65] Marcelo Fantinato. A feature-based approach to web services e-contract establishment. Technical report, Institute of Computing, University of Campinas, Brazil, 2007.

[66] William A. Hetrick, Charles W. Krueger, and Joseph G. Moore. Incremental return on incremental investment: Engenio's transition to software product line practice. Technical report, Engenio Storage Group and BigLever Software, 2006.

[67] Inc. ICON Group International. *Academicals: Webster's Quotations, Facts and Phrases*. ICON Group International, Inc., 2008.

[68] Ho-Won Jung, Seung-Gweon Kim, and Chang-Shin Chung. Measuring software product quality: A survey of iso/iec 9126. *IEEE Software*, 21(5):88–92, 2004.

[69] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–oriented domain analysis (foda) feasibility study. Technical report, Software Engineering Institute, Carnegie Mellon, 1990.

[70] Chris Karakas and Claudio Erba. *PHP-Nuke: Management and Programming*. Karakas-online, 2003.

[71] Charles W. Krueger. Software product lines.

[72] Alexei Lapouchnian. Goal-oriented requirements engineering: An overview of the current research. Technical report, University Of Toronto - Department of Computer Science, 2005.

[73] Kwanwoo Lee, Kyo C. Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. In *Software Reuse: Methods, Techniques, and Tools: Proceedings of the Seventh Reuse Conference (ICSR7)*, pages 62–77. Springer-Verlag, 2002.

[74] Yogev Lidor. Creating a wizard in visual composer for sap netweaver composition environment. Technical report, SAP, 2008.

[75] Xin Liu. Generating uml diagrams using feature diagrams for software product line. Master's thesis, Technical University of Eindhoven, 2006.

[76] Robert C. Martin. Uml tutorial: part 1 class diagrams.

[77] Varvana Myllärniemi, Mikko Raatikainen, and Tomi Männistö. Kumbang tools. Technical report, Helsinki University of Technology, 2007.

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

[79] pure-systems GmbH. Pure::variants user's guide, 2008.

[80] pure-systems GmbH. *Pure::variants User's Guide: Version 3.0 for pure::variants 3.0*, 2008.

[81] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with uml multiplicities. In *Design & Process Technology (IDPT2002)*, 2002.

[82] Matthias Riebisch. Towards a more precise definition of feature models. Technical report, Technical University Ilmenau, Germany, 2006.

[83] Germain Saval. Glossary of conference management, December 2006.

[84] Danilo Scalet, Alexandre Yokohama, André Koscianski, Claudete Maria Rêgo, Cleusa Asanome, Dantom Romero, Jeanine M. Cieslak, Marco Paludo, Ronaldo S. Frossard, and Tânia Mara Vostupal. Iso/iec 9126 and 14598 integration aspects: A brazilian viewpoint. In *The Second World Congress on Software Quality*, 2000.

[85] Effexis Software. Uml sequence diagram tutorial.

[86] Klaus Svarre. Content management system, 2006. http://searchsoa.techtarget.com/sDefinition/0,,sid26gci508916,00.html.

[87] M. Tseng and J. Jiao. Handbook of industrial engineering: Technology and operations management, 2001.

[88] H. Unphon. A comparison of variability and configuration tools for product line architecture. Technical report, IT University of Copenhagen, 2008.

[89] Erik van Veenendaal. Software testing glossary.

[90] Thomas von der Maßen and Horst Lichter. Requiline: A requirements engineering tool for software product lines. Technical report, RWTH Aachen, 2004.

[91] Karl E. Wiegers. *Software Requirements, 2nd Edition.* Microsoft Press, 2003.

[92] Steve Williams. What is a content management system, or cms?, 2008. http://www.contentmanager.eu.com/history.htm.

[93] Jeff Witkowski. Cms - content management system. http://knol.google.com/k/jeff-witkowski/cms-content-management-system/3uo0bldfjvjek/2.

[94] Tewfik Ziadi, Jean-Marc Jézéquel, and Frédéric Fondement. Product line engineering with uml. Technical report, IRISA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France, 2005.

Index