CAPTURING REQUIREMENTS VARIABILITY INTO COMPONENTS

A goal driven approach

Sondes Bennasri, Carine Souveyet
Centre de recherche en informalique, 90 rue de Tolbiac, 75013 paris, France

Keywords: Variability, Customisable Software, Requirement Engineering, Component

Abstract : Software Customisation also known as Software Variability is a central concept in the development of different kinds of software such as product families or software for disabled people. The solutions proposed in the literature to deal with the variability address design and implementation aspects like the mechanisms that can be used to implement the variability in a software architecture. The representation of the variability at a requirements level is neglected. Our contribution in this paper is to propose a goal driven approach that captures the variability at requirements level and maps it into a component-based solution centred on the concept of Customisable Component. An identification process is provided to assist the designer during the identification and the conceptualisation of the customisable components. The approach is illustrated with the Crews L'Ecritoire software.

1 INTRODUCTION

Today software companies are faced with the challenge of producing software systems that meet the needs of different kinds of users given the fact that at the same time they must decrease their costs. Thus, software should be sufficiently generic to cover a wide range of customer needs, easily adaptable to the requirements of a particular user and based on the reuse of existing software assets to reduce costs. This fact leads to the emergence of *software customisation* (also called *software variability*) which is defined as the ability of a software system to be changed, customised or configured to a specific context (Van Curp, 2000). Users of this kind of software play a key role as the success of the software depends on its ability to meet the user specific needs. Thus, user requirements should be considered at the first place during the process of software customisation and also when designing the customisable software.

In the first case, the customer is faced to a multitude of variants, he needs a global view of what each variant does and its dependencies with other variants without being lost in technical details. A representation of the variants at the requirements level facilitates the matching between his requirements and the software functionality.

In the second case, identifying the variability at the requirements level, assures that the designer is building a product satisfying user needs and provides a systematic way to document design alternatives.

Unfortunately, as mentioned in (Halmans et al., 2003) the representation of the variability at the requirements level is neglected. In general, the existing approaches such as (Bachmann et al., 2001)(Bosch et al., 2001)(Svahnberg et al., 2001) study the variability as a design problem and concentrate on implementation aspects of system variability.

We propose an approach that treats the variability from a requirements perspective. In this paper, we limit ourselves to variability in functionality. To identify the functionality variants, we propose to use a goal-driven modelling formalism called *Map* (Rolland, 2000) to capture the variability through requirements analysis and to map the variants into software components.

The choice of a component based solution to implement the variability is motivated by the desire to avoid reinventing the wheel every time a new system is developed but to package functions into reusable blocks that can be simply and straightforwardly integrated into new applications.

Bennasri S. and Souveyet C. (2004).
CAPTURING REQUIREMENTS VARIABILITY INTO COMPONENTS - A goal driven approach.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 438-443
DOI: 10.5220/0002638104380443

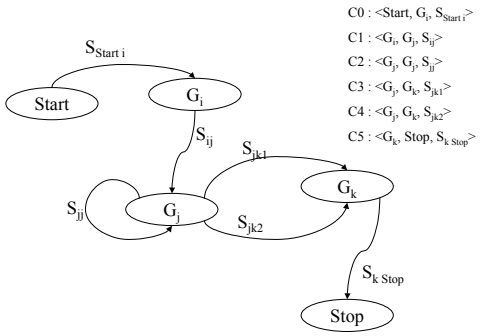

Figure 1 : A map

The reminder of the paper is structured as follows. Section 2 introduces the map formalism and the example that we choose to illustrate our approach. In section 3, we discuss how the map is used to represent the variability in functionality and how the variants are mapped into components. Finally, we draw some conclusions.

2 THE MAP FORMALISM

Our work is an extension of previous research results for matching ERP functionality to customer requirements (Rolland, 2000). We use the map to capture the variability at requirements level and implement them as software components.

A map is a process model expressed in a goal driven perspective. It provides a system representation based on a non-deterministic ordering of goals and strategies. In the next sub-sections, we introduce the key concepts of the map and we present the example that we use to illustrate our approach.

2.1 Map Concepts

A map is a labelled directed graph (see Figure 1) with goals as nodes and strategies as edges between goals. The directed nature of the graph shows which goals can follow which one.

A *Goal* can be achieved by the performance of a process. Each map has two special goals, *Start* and *Stop* to start and end the process respectively.

A *Strategy* is an approach, a manner to achieve a goal. The strategy S_{ij} characterises the flow from the source goal G_i to the target goal G_j and the way G_j can be achieved once G_i has been satisfied.

A *Section* is the key element of a map. It is a triplet $<G_i, G_j, S_{ij}>$ and represents a way to achieve the target goal G_j from the source goal G_i following the strategy S_{ij}. Each section of the map captures the situation needed to achieve a goal and the specific manner in which the process associated with the target goal can be performed.

The sections of the map may be connected to each others when :

- a goal is achieved with different strategies. This is represented in the map by several sections between a couple of goals. Such a map topology is called a *multi-thread*.
- a goal can be achieved by the combination of different strategies. This is represented in the map by a couple of goals connected by several sequences of sections. Such a topology is called a *multi-path*. In general, a map from its *Start* to its *Stop* goals is a multi-path and may contain multi-threads.

As an example, consider the map of Figure 1, we depict six sections C0 to C5. C3 and C4 form together a multi-thread whereas {C1, C3} and {C1, C4} are two paths between G_i and G_k that form a multi-path.

2.2 An example

To illustrate our approach, we choose the sample of the Crews L'Ecritoire system which is dedicated to requirements elicitation (Tawbi, 2001). The map of Figure 2 represents the functional requirements that the system must fulfil to elicit requirements at a high level by goals and strategies. Each section in the map represents a requirement that the system must satisfy. A functionality (depicted by Ci in the figure) is attached to each section in the map to achieve the related requirement.

As we can notice, the section allows a direct coupling between a functional requirement and the functionality to satisfy it. It is the means by which we derive system functionality from functional requirements.

In the reminder of the paper, we reference the sections and their attached functionality by Ci. The map is composed of four goals namely "Discover goal", "Conceptualise goal"and

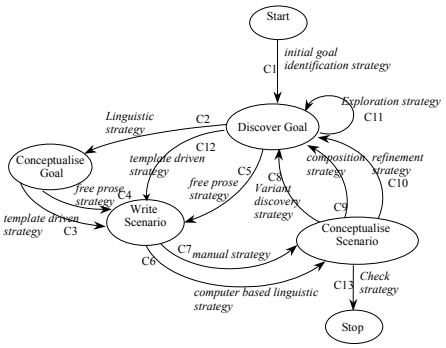

Figure 2 : The Crews L'Ecritoire map

"Conceptualise scenario" which represent the four steps that are followed to define requirements in Crews L'Ecritoire. The ordering of the goals reflects the principle of the bi-directional coupling of goals and scenarios in Crews L'Ecritoire. Once a goal is discovered, a scenario can be written to operationalise the behaviour of the system. We can also discover new goals from a scenario. Thus, the goal "Discover goal" precedes the goal "Write scenario". However to discover goals from a scenario, the scenario must be conceptualised that means it is written in a particular form allowing to automatically identify goals. This explains that the goal "Conceptualise goal" precedes the goal "Discover goal". Similarly, before writing a scenario, we may want to check the correctness of the goal through a linguistic analysis that reformulates its narrative description into a well structured form in order to align the scenario to the goal.

We shall notice different strategies from one goal to another that depict different manners to fulfil a goal. For example, there are two strategies to "Write a scenario" either in free prose (C5) or by filling a template (C12).

We can understand from this example, that the variability in requirements is captured through the different strategies proposed to satisfy the same goal. Further details about the variability in maps are provided in the next section.

3 VARIABILITY IN MAPS

3.1 Overview

We identify two kinds of variability in a map :

(i) a variability in the strategies used to fulfil a goal

(ii) a variability in the combination of the strategies to satisfy a goal

The first kind (i) is expressed by multi-thread topology. It shows through the strategies the different functionality provided to obtain the same result. For example, the multi-thread composed of {C12, C5} in Figure 2 depicts two alternative functionality to write a scenario from a goal, either by filling a template (C12) or in free prose (C5).

The second kind of variability (ii) is represented by the multi-path topology. It shows to users the several combination of functionality that they can execute to satisfy their needs. For example, a user interested in knowing how he can conceptualise a scenario from a goal has several paths between the couple of goals <Discover goal, Conceptualise goal>. The user can conceptualise his goal (C2), writes his scenario by selecting C4 or C3 then conceptualises it through C6 or C7. He can also decide to directly write the scenario by choosing C12 or C5 and then conceptualising it based on C6 or C7. The first multi-path {C2, C4, C3, C6, C7} is suitable when the user wants to check first the correctness of his goal before writing the scenario in order to ensure the adequacy of the scenario to the goal whereas the second multi-path {C12, C5, C6, C7} is followed when the goal is well written. If a user selects the second multi-path, he has many alternatives to write and conceptualise his scenario. He can select one of the paths {C12, C6}, {C5, C6}, {C12, C7} or {C5, C7}.

As we notice, the user is confronted with a multitude of alternatives. He can choose the best combination of alternatives according to his level of expertise in scenario writing.

For example, the path {C12, C6} is the best alternative for a beginner that needs assistance during the writing and the conceptualisation of the scenario. The path {C5, C6} also addresses the needs of beginners with less help during the scenario writing. The paths {C12, C7} and {C5, C7} are suitable for expert users.

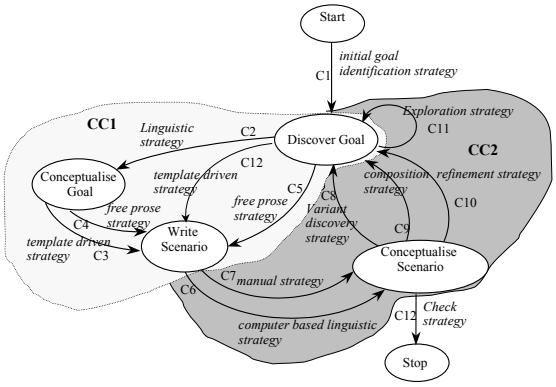

Figure 3 : Identification of customisable components

To sum up, the map represents the variability by using the multi-thread and multi-path topologies which describe the alternative functionality to satisfy user needs at a high level based on goals and strategies. However, representing the variability in requirements is one part of the work, we need also to document the variants such as the kind of users that is suitable for a particular variant. The information about the variants is captured by a document attached to the map.

3.2 Identifying components

In order to bring the variability captured at requirements level to the operational level, we introduce the concept of "Customisable Component".

A Customisable Component (CC) is composed of a set of functionality that operationalise a set of sections expressing the variability by multi-threads and multi-paths topologies leading to the achievement of a goal.

A CC also contains the information needed during its customisation according to specific needs such as the rationale of a particular variant.

The CC may be implemented by an assembly of existing software components or from scratch. In the second case, the CC is an abstract specification of functionality from which the designer can derive software component-based solutions.

The issue related to the implementation of the CC by software components is out of the scope of this paper. We concentrate only on the capture of the variability at requirements level and its operationalisation via the CC.

In the example described in section 2.2, we identify two CCs which are represented in Figure 3. The process leading to their identification from the map is explained in section 3.3.

The CC1 proposes two alternative ways to write a scenario either through the conceptualisation of the goal or directly by using a template or in free prose. The CC2 captures the alternatives to discover goals from a scenario. Three strategies are provided : by refinement strategy (C10), by composition strategy (C9) or by variant discovery strategy (C8). The first strategy is used to discover goals by considering the actions of the scenario as goals at a lower level. The purpose of the user is to refine his goals into system functionality. The second strategy aims to check the completeness of the requirements specifications by identifying the complementary goals and then writing their related scenarios. The last strategy discovers the alternative goals. It is useful to identify exceptional scenarios.

In the next sub-sections, we show how we identify the CCs from the map.

3.3 Process for identifying customisable components

The identification of the CCs follows two steps

(i) Identification of the candidate goals supporting the variability

(ii) Identification of the variants between each couple of candidate goals

(i) Identification of candidate goals

We identify the goals that are important for the user. We call them *candidate goals*. In our example, the most important goals are "Discover goal" and "Write scenario". The goals "Conceptualise goal" and " Conceptualise scenario" are intermediary ones that participate in the fulfilment of the important goals.

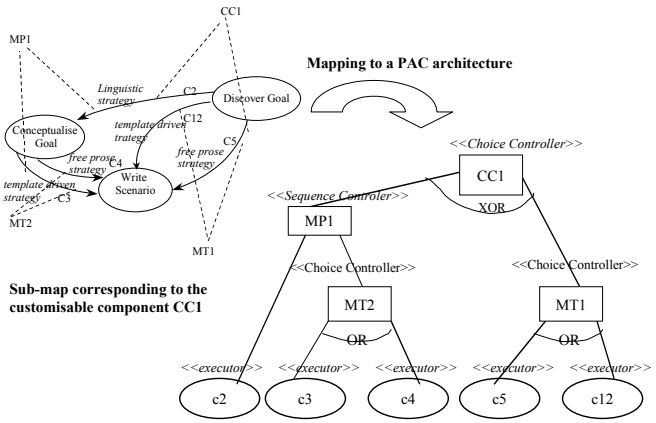

Figure 4 : Mapping a customisable component into a PAC like architecture

One heuristic to find the important goals in the map is to identify the relevant states of the products that the user wishes to obtain. In our example, the user is interested in having goals and scenarios which are the results of the fulfilment of the goals "Discover goal" and "Write scenario". Thus, the goals "Discover goal" and " Write scenario" are candidate ones.

(ii) Identification of the variability between couples of candidate goals

After identifying the candidate goals, we focus on the dependencies between them. In our example, we notice that in order to write a scenario we have to discover a goal first. Thus, the goal "Discover goal" must be realised before the goal "Write a scenario". We can also discover a goal from a scenario. In this case, we have to satisfy the goal "Write a scenario" before the goal "Discover a goal". We identify two couples of candidate goals that are: <Discover goal, Write scenario> and <Write scenario, Discover goal>. Each couple of goals is composed of a set of alternatives. We associate a CC to each couple of candidate goals.
We obtain the two CCs represented in Figure 3.
Once the customisable components are identified, we conceptualise them and organise them into a component architecture

3.4 Customisable component architecture

The CC is mapped to a component architecture which is a refinement of the Presentation-Abstraction-control (PAC) architecture (Buschmann et al., 1996).

The PAC architecture structures an application into a hierarchy of agents. Every agent consists of three components : presentation, abstraction and control. This subdivision separates the human-computer interaction aspect (encapsulated in the presentation component) of the agent from its functional core (the abstraction component) and its communication with the other agents (the control component).
We find that the PAC architecture allows to support the variability at an architectural level by defining the variants as modular entities that can be composed into several ways according to user needs. Moreover, the PAC architecture facilitates the evolution of its sub-elements. The evolution concerns :

(a) the agents: they are easily identified thanks to the precise role affected to them in the architecture
(b) the reorganisation of the agents and the introduction of new ones: in this case, the simplicity of the interaction schema between the agents facilitates the evolution.

To map a CC into a PAC architecture, we introduce two kinds of agents : the controllers and the executors
An executor is a self-contained semantic unit that provides a functionality.
A controller co-ordinates the lower-level agents that may be executors or controllers. We distinguish two kinds of controllers : the *choice controllers* that control the selection of the suitable child agent and the *sequence controllers* that manage the sequential execution of their children. Figure 4 shows how a CC is structured within a PAC architecture.
Each section Ci in the CC is mapped to an executor having the same name. It represents a functionality variant.

The multi-threads and multi-paths are managed by controllers. We identify three choice controllers MT1, MT2 and CC1 corresponding respectively to the multi-threads {C12, C5}, {C4, C3} and the multi-path embodying the two alternatives paths {C2, C3, C4} and {C12, C5}.
A sequence controller, identified in Figure 4 by MP1 is added to manage the multi-path composed of {C2, C3, C4}. Once the PAC architecture is defined, the interfaces of the controllers and the executors are specified. The PAC hierarchy is then implemented by a set of software components.

4 CONCLUSION

In our work, we use the requirements analysis as an input to capture requirements variability and to derive a component-based solution.
In our approach, we propose to capture the variability at the requirements level using the multi-thread and the multi-path topologies. The former shows the different manners to satisfy the same goal whereas the later captures the alternative paths leading to the satisfaction of a goal.
The variability identified at the requirements level is operationalised by a direct coupling between a requirement (a section of a map) and a functionality to achieve it.

Our approach is also driven by the component paradigm, we introduce the concept of Customisable Component that is a conceptual concept to describe the variability captured at the requirements level, at a conceptual level and to derive software component based solutions.

The originality of our work consists of dealing with variability from a requirements perspective. However, our approach is a work in progress.
In the literature, there are few works addressing the variability from a requirements perspective. We distinguish the proposal of (Hui et al., 2003) where the variability is captured through goal analysis using the AND/OR decompositions. The alternative goals (expressed by the OR links) help reasoning about the alternative functionality to achieve a parent goal. However, the exploration of the alternative combinations of functionality across the entire AND/OR goal graph is more difficult.
We find that maps, as means for describing alternative complex assembly of functionality, can help in this exploration. The multi-thread topology of maps corresponds to the OR link in a goal graph. In addition, the multi-path topology helps reasoning about the alternative assemblies of functionality.

Our future work consists in (a) detailing the rules to map a customisable component into a set of software components and (b) providing the approach to build systems using the customisable components.

REFERENCES

Bachmann et al., 2001. Managing variability in software architecture. ACM Press, NY, USA, 2001.

Bosch et al., 2001. Variability issues in Software Product Lines. 4th International Workshop on Product Family Engineering (PEE-4), Bilbao, Spain, 2001.

Buschmann et al., 1996. A system of patterns. Pattern-oriented software architecture. Wiley, 1996.

Halmans et al., 2003. Communicating the variability of a software product family to customers. Software and System Modeling, Springer-Verlag 2003.

Hui et al., 2003. Requirements Analysis for Customizable software : A Goal-Skills-Preferences Framework. 11th International Requirement Engineering Conference, 2003.

Rolland C., 2000. Bridging the gap between Organizational needs and ERP functionality. Requirements Engineering journal, 2000.

Svahnberg et al., 2001. On the notion of variability in Software Product Lines. Proceedings of the Working IEEE/IFIP Conference on Software architecture, 2001.

Tawbi M.,2001. Crews L'Ecritoire : un guidage outillé du processus d'Igenierie des besoins, PHD thesis, Paris1, 2001.

Van Gurp J., 2000. Variability in Software Systems, the key to Software Reuse. Licentiate Thesis, University of Groningen, Sweden, 2000.