From a Business Component to a Functional Component using a Multi-View Variability Modelling

Rajaa Saidi [1,2,3], Agnès Front [1], Dominique Rieu [1], Mounia Fredj [2], Salma Mouline[3]

(1) LIG – SIGMA Team, BP 72, 38402 Saint Martin d'Hères Cedex, France
{First name. Name}@imag.fr
(2) L@GI, Al Qualsadi, ENSIAS, BP 713, Rabat, Morocco
fredj@ensias.ma
(3) GSCM_LRIT, Mohammed V-Agdal University, B.P 1014, Rabat, Morocco
mouline@fsr.ac.ma

Abstract. The ability of reusable components to be varied and appropriate to different designers and re-users requirements is a key property in reusable component development, especially in Business Component (BC) development. The primary objective of this work[1] is BCs reuse in different Business Domains. In order to achieve this goal, we focus on the variability concept, which is defined as the ability of a software artefact to be changed or customized so as to be reused in multiple contexts. Thus, we introduce the concept of Functional Component (FC) that captures similarities and variations between BCs that share common behaviours in order to increase their reusability in different business domains. The FC is modelled by using a multi-view variability.

Keywords: Business Component, Functional Component, Reuse, Variability.

1 Introduction

Component reuse is defined as the process of implementing or updating information systems using existing assets, and the ability of reusable components to be varied and appropriate to different designers and re-users needs is a key property in reusable components development.

This process has been supported by research contributions from various fields, including generic data models [1], generic components [2], and domain model [3]. Most of this work aims at facilitating the reuse of generic aspects of a domain.

However, we presume that is also beneficial the reuse of variable development artefacts. For this, we focus on the concept of variability, which is defined as "the ability of a software artefact to be changed or customized to be used in multiple contexts" [4]. Variability was introduced into various contexts, particularly in domain engineering [4] and product lines development [5], [6] and [7]. However, it was

[1] This work is supported by the COMPUS Project : MA/06/151, PAI: VOLUBILIS 2006

expressed a little in reusable component design, especially in **Business Component** (BC) [8] development.

According to [9] and [10], a BC is a representation of an active concept in a business domain. Thus, BCs are used to define concepts (e.g. Patient, Hospital…) in a standard way or to define processes for the concepts which they represent (e.g. the Billing/Payment Process, the Medical Encounter Process…).

Researchers have proposed different structures of BC (see [8] and [9]). In this paper, we use the Symphony [11] conceptual model developed in our research team. Thus, a BC is a structure based on the CRC technique (Class-Responsibility-Collaboration) [12]. A BC is modelled as a package composed of three parts (cf. Fig.1):

- The **"Interface"** part describes what a *BC can do*. It defines the BC *services* which correspond to the operations representing use cases assigned to the BC (e.g. *ReserveBed()* is a part of the interface of the Hotel BC).
- The **"Master"** part describes what the *BC is*. It corresponds to the *structural* part (e.g. "*Hotel*" is a structural part). Yet, the class **"Part"** ("*Room"*) supplements the Master part to which it is connected by a composition relationship. It corresponds to a partial structuring of the attributes of the Master part, thereby highlighting a business aspect of the component.
- The **"Role"** part describes what a *BC uses*. It highlights the *collaboration* played with BC suppliers (e.g. "*RoomRes*" formalizes the contract between the class "*Room*" and a supplier (other BC interface, for instance the Reservation BC), thus, *"RoomRes"* class is described as a role).

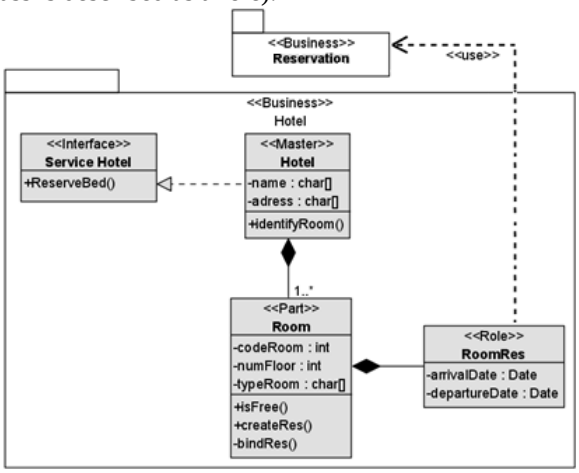

Fig. 1. Example of a BC

However, in this paper a special interest is given to the reuse activity in several business domains. Therefore, we propose to represent the BC at a generic level and to express all its variabilities. For this purpose, we introduce the concept of **Functional Component** (FC) that captures similarities and variations between BCs that share common behaviours in order to increase their reusability in different business domains. In this paper, our objective is to design an FC and express variability in

order to support the development and the reuse of variable development artefacts. Variability can be applied and occurs at different views of development (functional, dynamic and static). However, locating where an FC varies and the way we can realize its variations are complex tasks. Thus, with FC development, the necessity to trace variability between solution and problem spaces is inevitable, and the approaches dealing with this complexity of variability need to be established clearly.

This work aims to define the main concepts of variability for reuse, namely the requirements for identifying variability, thus outlining the crucial factor for determining the overall benefit of variability for reuse, especially how it can enrich a reuse space and improve the reusability of the developed BC. In this paper, to illustrate how to get an FC from similar BCs, we use an example which shows variability between two BCs from the same functional domain (Reservation), but from two different business domains: Tourism (Hotel Reservation) and Health (Hospital Reservation).

The remainder of this paper is structured as follows: in Section 2, we define the relations between a BC and an FC. Variability finality and capture concepts are illustrated in Section 3. In Section 4, we introduce an approach to represent a multi-view variability in an FC. Related works are presented in Section 5. Concluding remarks are given in Section 6.

2 From a Business Component to a Functional Component

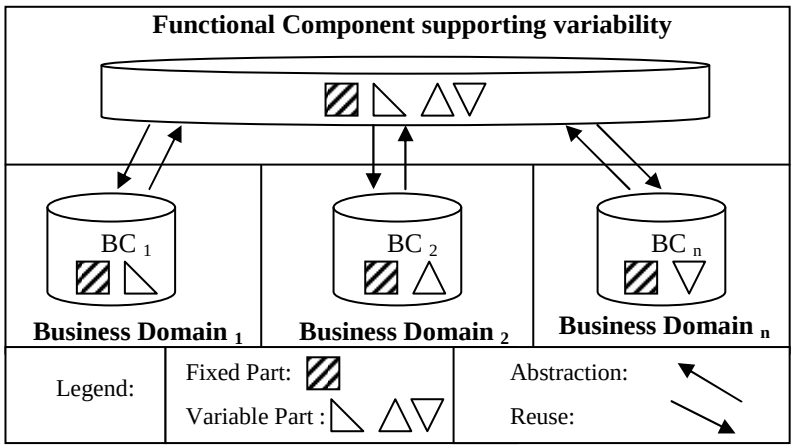

Fig. 2. Variability mechanism

In this paper, the purpose of our approach is the design of FCs that represent abstract recurrent behaviours (e.g. Resource Allocation, Recording…) among several business domains (e.g. Tourism, Health…), in order to reuse them in other business areas (e.g. Transport). The main idea here is to abstract BCs that share these behaviours related to their own business domains. This consists of the identification of what is common

and what is variable between these BCs. Hence, using variability mechanism is needed.

To support the variability concept, we propose that the specification of an FC has to distinguish between a fixed part and variable parts (cf. Fig.2). The fixed part represents the component structures which are reusable in each business domain that requires these structures; they are directly integrated in the BC under construction. The variable parts represent the component structures for which reuse requires a selection process adapted to a particular business domain.

Variability in FC is then defined as a process which is used to control the common and variable parts of FC artefacts. Variability resolution (exploitation) produces a collection of particular FC reuse cases. Yet, the abstraction (design for reuse) is a process that identifies reusable artefacts from BCs that already exist in several business domains, and the reuse (design by reuse) is a process that produces reuse cases of FC artefacts in order to develop BCs. Both processes are based on the variability technique.

Yet, a BC can be also reused in several applications within the same business domain. Thus, BC reuse gives a set of Business Objects which are modular structures specific to a given information system [11]. Business objects can represent entities (customer, supplier…) or processes (order, invoice…) associated to an information system. This type of reuse is not undertaken in our paper.

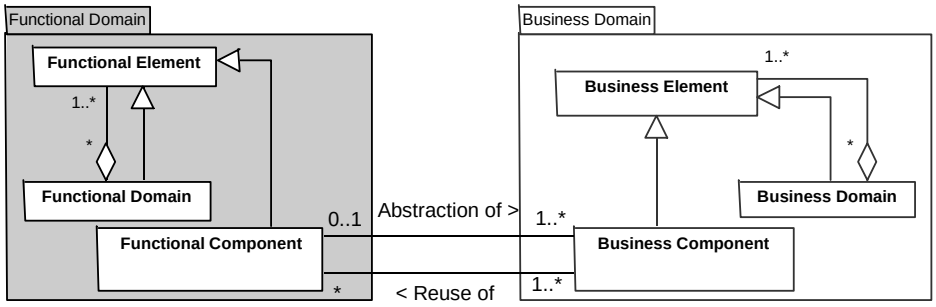

Fig. 3. FC and BC metamodel

The metamodel depicted in Fig.3 highlights the relation between a BC and an FC. The FC represents the central concept of a functional domain. A functional domain is composed of functional elements that correspond to knowledge which is independent from any business area whereas a BC represents the central concept of a business domain which is composed of business elements that correspond to business knowledge. An FC is built from the abstraction of at least one BC, and a BC can be built by reusing at least one FC or by the abstraction of at least one business objet of the same business domain. The relation between BC and business object is not represented in our metamodel.

Figure 4 depicts an example of functional domains composed of FCs and business domains composed of BCs. The “Resource Allocation” can be made with or without reservation, thus, we represent it with the FC1 which can be specialised by both FC2 and FC3. To illustrate the relation between FCs and BCs, the example shows that FC3

can be reused in several business domains, so it can be reused to design BC1, BC2 or BC3. In addition BC1, BC2 and BC3 can be abstracted to design FC3.

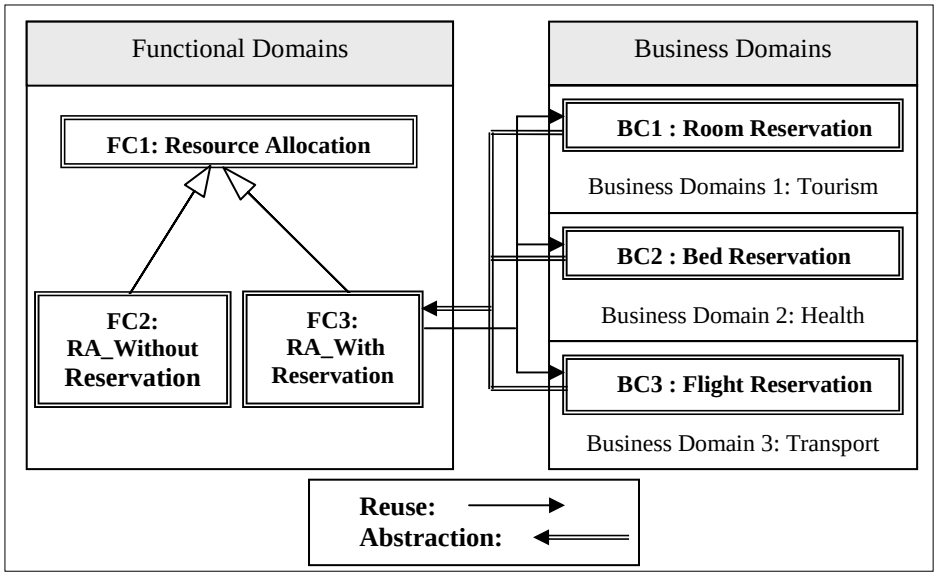

Fig. 4. Example of FCs and BCs

3 Variability finality and capture

There are many concepts involved in variability management. In the present section, we introduce several concepts that we consider as the minimum necessary to specify an FC supporting variability. These concepts deal with tow points: variability finality and variability capture.

3.1 Variability finality

Variability finality outlines the scope of solutions being developed according to the variability concept. Thus, we state that it can be applied according to two views: variability for *customization* and variability for *reuse*:

- Variability for customization holds when BC solutions are to be customised by other developers into user solutions. These developers can configure and instantiate prespecified BC to express their variability.
- Variability for reuse consists of making variability explicit during the design for reuse process; this means that an FC specification defines the variable parts and the fixed part of this FC in order to increase its reusability and applicability in different business domains.

Yet, variability comes from variable parts of the BCs in the domain which can vary in many ways:

1. One BC implements a feature in several ways and offers the user a choice among them.
2. Several BCs from different business domains share the same feature but implement it in different ways and it is foreseeable that, in future BCs, the feature will be implemented in other ways.

In our case, in which we want to express variability for reuse, we focus on the second point. This case shows that variability can be identified in several BCs and can be predictable for future BCs.

3.2 Variability capture

In the initial phase of the FC development, we are confronted to requirements that allow us to identify where variability is needed. Thus, variability capture is the most important phase to discover where BC varies.

In our context, for variability capture, we propose to use a list of BCs within the same family of systems i.e. systems that share common characteristics. Then, we proceed to deal with the commonalities and differences between those BCs. Hence, we use in this paper two similar BCs of the "Reservation" system and we try to represent them with an abstract BC as an FC. The first BC is the "Bed Reservation" in a "Hospital" business domain and the second BC is the "Room Reservation" in the "Hotel" business domain. Commonalities are the fixed parts of the FC, and differences are the variable parts of the FC, which are the parts that make the BCs distinct from one another. Yet, for more details about variations, we propose that the specification of each BC must be at different views (i.e. functional, dynamic and static). Due to space limitations, we cannot present a full account of the variability capture phase.

The abstraction activity aims to generate an FC by comparing similar BCs according to the three views. In the "Room Reservation" and the "Bed Reservation" BCs, for instance, the comparison of their functional and dynamic views leads to the generation of abstract artefacts by a simple use of reservation taxonomy. All we have to do is to replace "Room" and "Bed" by "Stay Place", "Secretary" and "Receptionist" by "Reservation Responsible", "Hotel" and "Hospital" by "Stay Provider", etc.

However, at the static view, the abstraction activity is more complex. This situation shows the presence of variation between both BCs. This variation is denoted by the way the "Stay provider" and the "Stay place" are related. In fact, in the "Hotel" BC, the "Room" class is directly related to the "Hotel" class. Nevertheless, in the "Hospital" BC, the "Bed" class is related to the "Hospital" class by an intermediate class, which is the "Room" class. This statement leads to the distinction of a **structural variability**, which occurs when there are classes related in several ways. Hence, we denote the differences identified between our two BCs by the degree of depth of related classes that vary from a BC to another. Once a variation is identified, it needs to be constrained. Thus, it is eventually designed as variable part; we have illustrated this point in the context of a precedent paper [13]. In the next section, we show how to represent this variability across the three views of the FC.

4 Multi-View Variability Modelling in an FC

As we mentioned in Section 3, the variation identified in our BCs is the degree of depth when a "Stay Place" is related to a "Stay Provider".

Hence, to represent differences between BCs of the same family, we use the concepts of *Variation Points* (VP) [14] and / or *Variants* (V) [15]. According to our context, we define VP and V as follows:

- Variation Point: represents a location on which a variation will occur. It is a conceptually common feature that can be implemented in different ways. Thus, it locates a specific place in an FC artefact to which a later decision is attached.
- Variant: represents a specific implementation of a variation point. It corresponds to design alternatives to resolve the variability.

Thus, in this example "Identify StayPlace" is considered as a Variation Point and "Identify StayPlace_0 degree" (i.e. a "Room Reservation"; the class "Room" is related directly to the "Hotel" class) and "Identify StayPlace_1 degree" (i.e. a "Bed Reservation"; the class "Bed" is related to the "Hospital" class by an intermediate class: "Room" class) are considered as Variants of this VP.

In this paper, to represent variability, we propose to use UML extension carried out by stereotypes. UML is a good candidate for variability representation, because there is wide range of tools supporting it, and the notation is extensible. Therefore, we will apply this approach on the functional, dynamic and static views of our FC.

4.1 Functional variability

At the functional view, our proposed stereotypes are supported by use cases in the use case diagram. Stereotypes that we use in our example are given as follows:

- « Variation »: to specify if a use case is a VP.
- « Variant »: to specify a variant related to a VP.
- « Alternative »: to specify the relation between a VP and its alternative Variants.

Figure 5 depicts a variable specification of the functional view of the FC. "Identify StayPlace" is a VP while "Identify StayPlace_0 Degree" and "Identify StayPlace_1 Degree" are its alternatives variants.

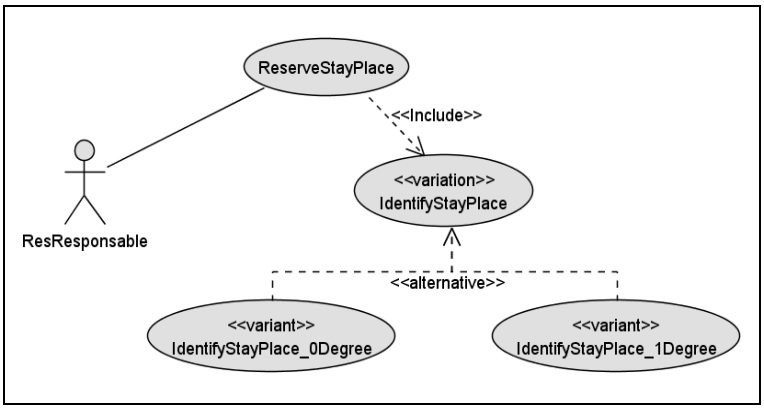

Fig. 5. Variability in the functional view of the FC

4.2 Dynamic variability

Expressing variability in the dynamic view is based on frames (fragments of interaction). Thus, we include sub-sequences corresponding to alternatives. This technique requires the application of the following rule: any Variation Point of the functional view must be represented by at least a frame of a sequence diagram in the dynamic view. Hence, we respectively use the stereotypes « Variation » and « Variant » on the fragments related to a VP and V.

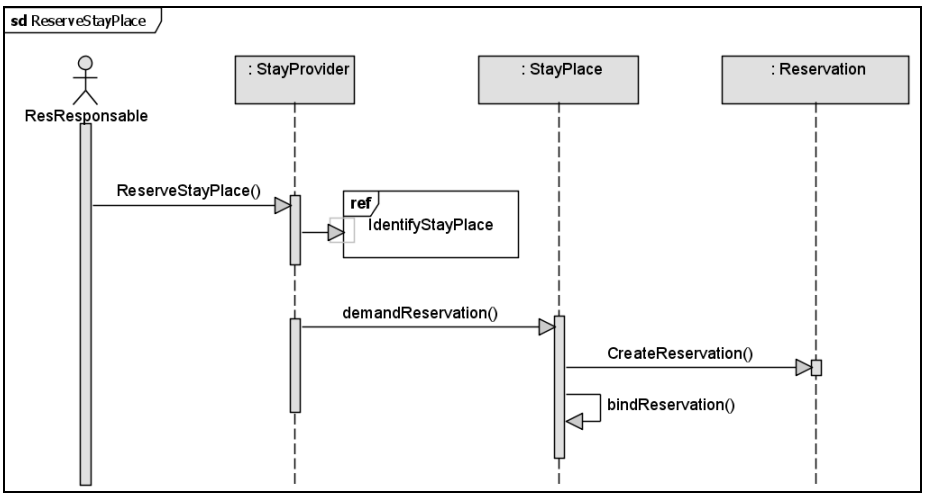

Fig. 6. Variation in the dynamic view of the FC

Figure 6 depicts a variable specification of the dynamic view in the FC. It illustrates the frame for the use case “Reserve StayPlace”, including a reference to the

"Identify StayPlace" fragment. Figure 7 shows how the dynamic parts of the variants are included in the variation point.

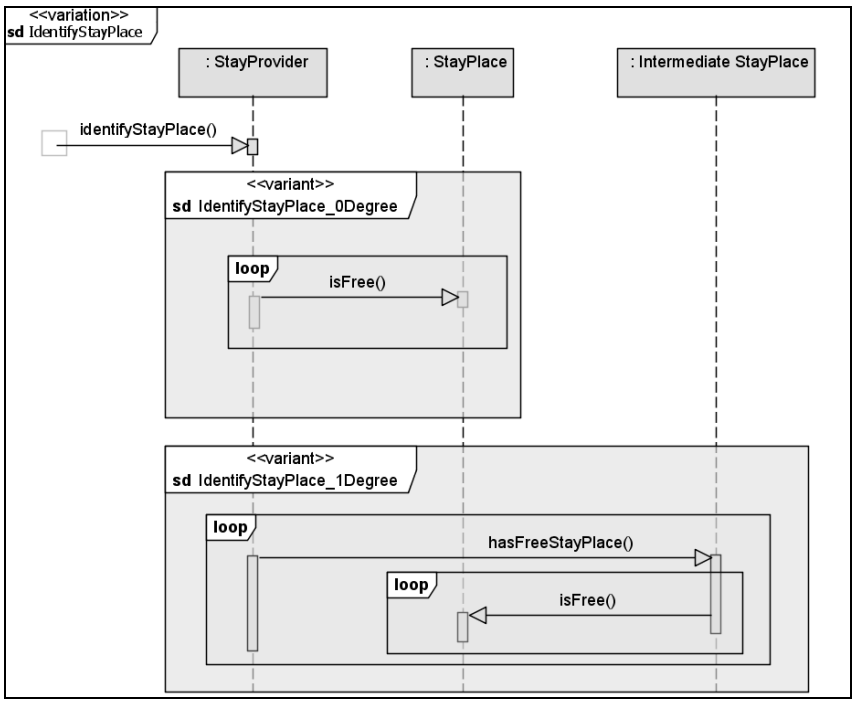

Fig. 7. Variants in the dynamic view of the FC

4.3 Static variability

This phase consists of the specification of the static contributions of each functionality to finalize our FC. Indeed, for each specified use case, we give the static contributions in the solution model.

The structure is distributed in several fragments which will be assembled to form a traditional static view at the reuse activity. Each class impacted by a variation or by a functionality will be represented in the static fragment of this use case. Figure 8 depicts static fragments of the use cases: Reserve StayPlace (8.a), Identify StayPlace_1 Degree (8.b) and Identify StayPlace_0 Degree (8.c).

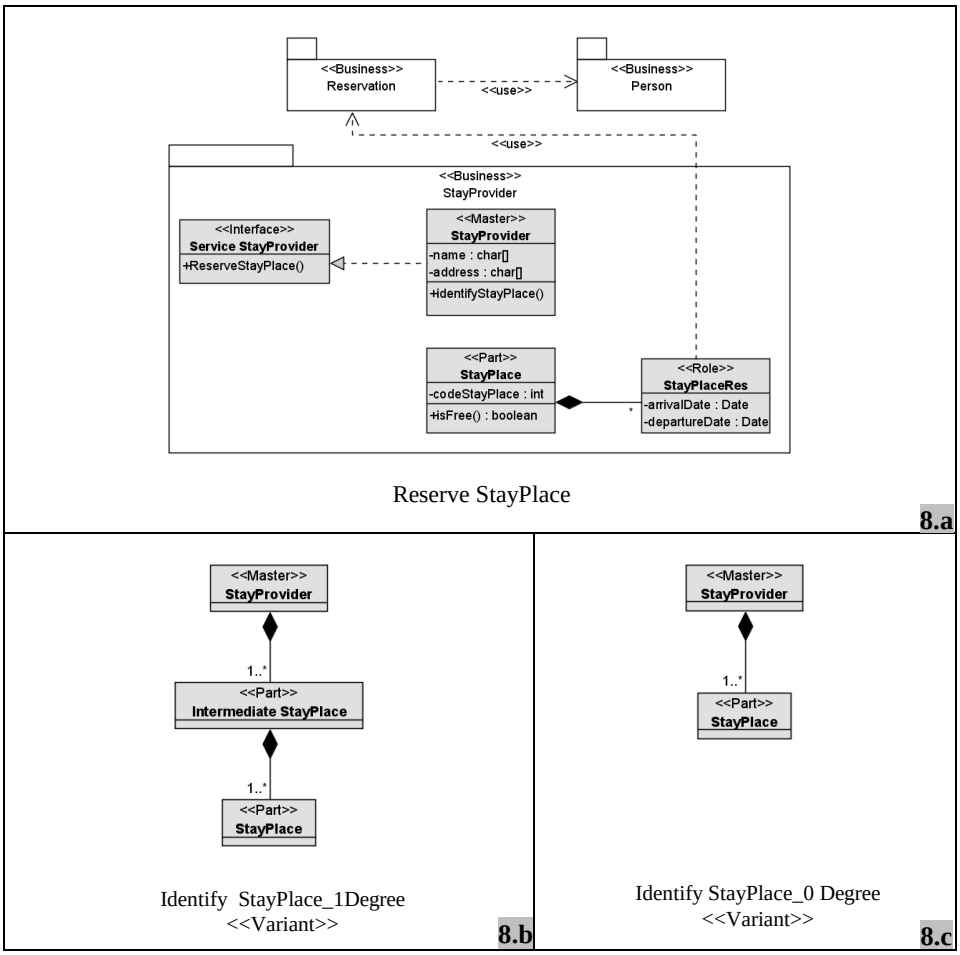

Fig. 8. Variability at the static view of the FC

Once variability is modelled across the three points of view, we obtain an FC supporting variability. According to our example, we also mention that it is worthwhile to note that having designed an FC allows many BCs to be derived in other business domains such as the "Transport" business domain. For example, we can reuse our FC for a "Train Place Reservation". This FC reduces software development efforts. In this context, we make use of a horizontal reuse across a wide variety of business domains.

Yet, our approach can also be applied for vertical reuse. In this case, the basic idea is the reuse of a BC supporting variability that can be used by a family of information systems with similar requirements in the same business domain. In this case, variability can be used with the customization finality.

5 Related works

Variability was originally introduced in the FODA method by the feature model [4]. It consists of a set of nodes, a set of directed edges, and a set of edge decorations. Common features are defined as mandatory features that will be included in all the member products in the product lines. Variable features are configurable features, i.e. they are not necessarily included in every derived product and can be selectable. Unfortunately, the applicability of this notation is limited by the fact that it requires special propose modelling tools.

Therefore, several extended UML methods have been proposed (see [14], [15] and [16]) in order to represent the variability concept. The extension is realised by using standardised extension mechanisms of UML by using stereotypes, and can be expressed as a UML profile [17].

In our work, we based on this solution. In fact, extending UML for modelling variability in BC artefacts, throughout different phases of development (requirements, analysis, design and implementation) or different points of view (functional, static and dynamic), has the benefit to guarantee traceability among corresponding artefacts, and the particularity of our work consists of taking into account the business aspects in order to represent this variability.

Furthermore, variability, as a separate model was proposed, and was denoted by Variability Orthogonal Model (VOM) in the product line context. The VOM, as it is described in [7], has two central elements, which are variation point and variant classes. A variation point represents an explicit engineering decision which provides several alternatives variants with regard to selected assets or artefacts of the system development. However, we perceive that the VOM is specific to product lines approaches and its finality is mainly customisation. Thus, it is not enough evaluated to fit the variability for BC reuse requirements, and therefore, it needs further empirical verification.

6 Conclusion

This paper described the potential of variability modelling through BC development. To design a generic BC, which supports variability, we have introduced the concept of FC which captures similarities and variations between BCs that share common behaviours. Subsequently, the FC can be reused in multiple business domains.

Our proposed approach needs further study; it means that after designing functional components supporting variability, it is essential to implement a resolution process of this variability, as well as the integration of these components in the development process of a components-based information system.

Therefore, we presume that only conceptual models of functional components, such as UML models, produced during the specification or design processes, are real reusable artefacts for the design of components based information systems; they are technologically neutral and evolve according of the changing needs for which they respond.

References

1. Mineau, G.W., Godin, R.: Automatic structuring of knowledge bases by conceptual clustering, IEEE Transactions (Data and Knowledge Engeering, 7 (5), pp.824-829, (1995)
2. Castano S., De Antonellis, V.: The F^3 Reuse Environment for Requirements Engineering, ACM SIGSOFT Software Engineering Notes, 19 (3), pp. 62-65, (1994)
3. Snoeck, M., Poels, G.: Analogical Reuse of Structural and Behavioural Aspects of Event-Based Object-Oriented Domain Models, Domain Engineering Workshop, Proceedings of the 11th International Workshop on Database and Expert Systems Applications, pp. 802-806, London (Greenwich), 4-8 Sept. (2000)
4. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) feasibility study:, Technical report CMU/SEI-90-TR-21, Software Engineering Insitute, Carnegie Mellon University, (1990)
5. Gurp, V., Bosch, J., Svahnberg, M.: Managing Variability in Software Product Lines, Landelijk Architectuur Congres, Amsterdam, (2000)
6. Bachmann, F., Bass, L.: Managing variability in software architecture, ACM SIGSOFT Software Engineering Notes, Volume 26, n°3, (2001)
7. Pohl, K., Bockle, G., Linden, F.V.D.: Software Product Line Engineering: Foundations, Principles, and Techniques, Springer-Verlag Berlin Heidelberg, (2005)
8. Barbier, F.: Business Component-Based Software Engineering, the Springer International Series in Engineering and Computer Science, Vol.705, 280 p, (2002)
9. Cummins, F.: OMG Business Object Concept – BOTF –, White paper – EDS- BOM/99-12-42, (1999)
10. Casanave, C.: Business Object Architectures and standards, Data Access Corporation, Miami USA, (1996)
11. Hassine, I., Rieu, D., Bounaas, F., Seghrouchni, O.: Towards a Reusable Business Components Model, OOIS02, Montpellier, (2002)
12. Wirfs-Brock, R., Wilkerson, B., Weiner, L.: Designing Object-Oriented Software Prentice-Hall, Englewood Cliffs, New Jersey, (1990)
13. Saidi, R., Fredj, M., Mouline, S., Front, A., Rieu, D.: Towards Managing Variability Across Business Component Development, Proceedings of the 2007 IEEE International Conference on Information Reuse and Integration, IRI – 2007, 13-15 august, Las Vegas, USA, (2007)
14. Clauss, M.: Generic modeling using UML extensions for variability, Workshop on Domain Specific Visual Languages, pages 11-18, (2001)
15. Oliveira, E.A., Gimenes, I., Huzita, E., Maldonado, J.C.: A Variability Management Process for Software Product Lines, In Proc. of CASCON 2005, Toronto, Canada, (2005)
16. Arnaud, N., Front, A., Rieu, D.: Expressing variability for patterns re-use, First International Conference on Research Challenges in Information Science, RCIS 2007, Ouarzazate, Morocco, April 23-26, (2007)
17. Ziadi, T., Hélouët, L., Jézéquel, J.M.: Towards a UML Profile for Software Product Lines, 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, (2003)