Product Lines' Feature-Oriented Engineering for Reuse: A Formal Approach

Marcel Fouda Ndjodo[1], Amougou Ngoumou[2]

[1] **Department of Computer Science and Instructional Technology, University of Yaoundé I**
P.O. Box 47 Yaoundé - Cameroon

[2] **Department of Computer Science, The University Institute of Technology, University of Douala**
P.O. Box 8698 Douala - Cameroon

Abstract

The feature oriented method with business component semantics (FORM/BCS) is an extension of the feature oriented reuse method (FORM) developed at Pohan University of Science and Technology in South Korea. It consists of two engineering processes: a horizontal engineering process driven by the FORM domain engineering process and a vertical engineering process driven by the FORM application engineering process. This paper investigates the horizontal engineering process - which consists of analyzing a product line and developing reusable architectures – and shows that this process can be systematized through a set of maps that describe how one can systematically and rigorously derive the fundamental business architectures of a product line from the feature model of that domain. The main result of the paper is therefore that the formalization of the assets of FORM/BCS enables a clear definition of how an activity of the horizontal engineering process produces a target asset from an input one. This result opens the door for the development of a tool supporting the method.

Keywords*: Product Line Engineering, Feature-Orientation, Domain Analysis, Business Components, Reuse, Formal Method.*

1. Introduction

FORM with Business Component Semantics [1], which is an extension of FORM [2, 3, 4, 5, and 14], is a feature-oriented product line engineering method. It has two processes: a horizontal engineering process driven by the FORM domain engineering process and a vertical engineering process driven by the FORM application engineering process. These two processes correspond respectively to the "engineering for reuse" and the "engineering by reuse" approaches presented in [6, 7, and 8].

The FORM/BCS method is specific to other product line engineering approaches [9, 10, 11, 12, and 13] by the fact that its vertical engineering process gives the possibility to successively derive concrete business components of successive low level application domains from abstract reusable business components of a high level domain. Since the vertical engineering process of FORM/BCS is an "engineering by reuse" approach, it can be applied only if the reusable business components of the initial domain already exist. These assets are produced by the horizontal engineering process of FORM/BCS, which consists of analyzing a product line domain and developing fundamental reusable architectures of the domain. The horizontal process is therefore the core process of FORM/BCS.

This work formalizes this process with a set of maps which describe how to rigorously derive the fundamental business architectures of a domain from the feature model of that domain. Each map specifies an activity of the horizontal engineering process by defining how that activity produces a target asset from an input one. It has been possible to formalize the horizontal process of FORM/BCS since one of its particularity is that its assets are defined using the Z notation which provides a framework for a rigorous analysis of the method.

This work is a significant contribution to the development of FORM/BCS since it lays down the theoretical foundation for the development of a tool supporting this method. Beyond the development of FORM/BCS, which is the main focus of the paper, this work addresses some key open issues of feature-oriented development. One of these issues is the automation of feature-oriented development processes. This work shows that formalization is an important step toward this goal.

The rest of the paper is organized as follows: Section 2 gives an overview of FORM/BCS. Section 3 presents the assets of the method. Section 4, which is the main part of the paper, formalizes the horizontal engineering process of FORM/BCS. Section 5 presents related works and gives the end

result of this article through a consistency theorem. Finally the conclusion presents possible further research domains in the work continuation.

2. Overview of FORM with Business Component Semantics.

2.1 The Horizontal Engineering process

The aim of the horizontal Engineering process is to analyze a domain in order to produce the reusable business components of that domain which consist of (i) a feature business component, (ii) a subsystem business component, (iii) a set of process business components and, (iv)a set of module business components.

The horizontal engineering process has four independent activities: the domain analysis activity, the subsystem architecture business component design activity, the process architecture business component design activity and the module architecture business component design activity.

1) Given a domain, the domain analysis activity, which is intuitive, produces a reusable feature business component for that domain. This feature model, defined in section 3.2, is stored in a database of reusable feature business components.
2) Given a reusable feature business component selected in the database of reusable feature business components, the subsystem architecture business component design activity produces a reusable subsystem architecture business component, defined in section 3.3, which is stored in a database of reusable subsystem architecture business components.
3) Given a reusable subsystem architecture business component, the process architecture business component design activity produces a set of reusable process architecture business components, defined in section 3.4. These components are stored in a database of reusable process architecture business components.
4) Given a reusable process architecture business component, the module architecture business component design activity produces a set of reusable module architecture business components, defined in section 3.5. These components are stored in a database of reusable module architecture business components.

The following properties of the FORM/BCS Domain Analysis process described above can be considered as the main improvements of the original FORM Domain Analysis process:

- Reusable assets integrate context: information about assets reuse in the form of context is integrated in assets.
- The implementation of the activities in the new method is not linear; that is, they can be performed in any order. The assets produced by activities of the method are stored in a database of reusable assets which can be requested using the operators for reuse developed by Ramadour [10]: search operators, selection operators, adaptation operators and composition operators.
- A database of business components is added: this database enables engineering for reuse through the storage of assets and engineering by reuse through requests which can be submitted to it using the Ramadour's operators for reuse listed above.

2.2 The Vertical Engineering process

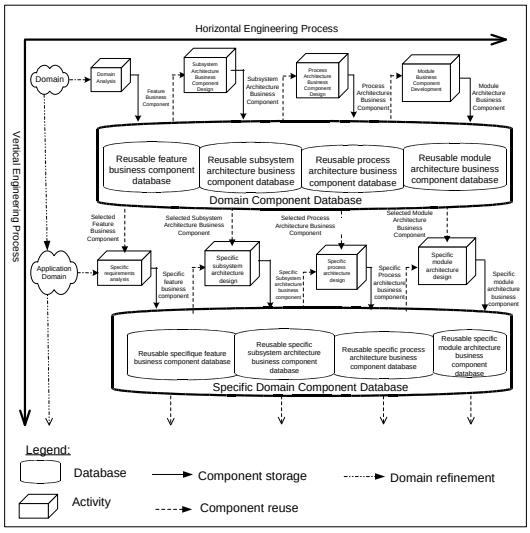
The aim of the Vertical Engineering process is to derive a database of reusable components of an application domain of a domain which already has a database of reusable domain components. The Vertical Engineering process has four independent activities: the specific requirements analysis, the specific subsystem architecture design, the specific process architecture design and the specific module architecture design.

1) Given an application domain A of a domain D and a feature business component F of D, the aim of the specific requirements analysis of the domain A is to derive a feature business component F' of A from F. For this, the activity operates choices in F to reduce the number of optional features or groups of alternative features contained in the decomposition of its solution. The derived feature business component F' is stored in a feature business component database of A.
2) Given a feature business component F' of an application domain A derived from a feature business component F of a domain D, and a subsystem architecture business component S produced from D, the aim of the specific subsystem architecture design activity is to derive a subsystem architecture business component S' of A from F' and S. For this reason, the activity eliminates features contained in sub systems of the subsystem architecture business component S, which are absent in the basic feature business component F'.
3) Given a subsystem architecture business component S' of an application domain A derived from a subsystem architecture business component S of a domain D, and a process architecture business component P produced from D, the aim of the specific process architecture design activity is to derive a process architecture business component P' of A from S' and P. For this, basing on S' the activity adapts the process architecture business component P.

4) Given a process architecture business component P' of an application domain A derived from a process architecture business component P of a domain D, and a module architecture business component M produced from P, the aim of the specific module architecture design activity is to derive a module architecture business component M' of A from P' and M. Here, basing on P', the activity adapts the module architecture business component M.

The possibility of successive refinements of reusable business components of a domain to more concrete components (vertical engineering) is the main improvement of the Application Engineering process of the original FORM.

The FORM/BCS engineering process is pictured as follows:



3. The Assets of FORM/BCS.

This section describes the four main assets of FORM/BCS: feature business components (section 3.2), subsystem architecture business components (section 3.3), process architecture business components (section 3.4) and module business components (section 3.5). Since all the assets of FORM/BCS are reusable components, section 3.1 presents the conceptual modeling of reusable business components which is used to define the FORM/BCS assets as reusable business components.

3.1 The formal model of FORM/BCS assets.

The assets of FORM/BCS are reusable business components which are stored in a database. Each reusable business component has a *name*, a *descriptor* and a *realization*. The descriptor presents the conceptual modeling problem to be solved in a particular context. This problem can be the decomposition of a system, an activity organization or an object description. Goals, activities and objects concerned are carried on an application field and/or an engineering method. The realization section of a reusable component provides a solution to the modeling problem expressed in the descriptor section of the component. This solution may have *adaptation points* with values are fixed at the reuse moment. Adaptation points enable the introduction of parameters in the solutions provided by reusable components. The following Z schema formalizes reusable business components:

ReusableBusinessComponent = =
[name: ***Text;***
*descriptor:****Descriptor***
realization: ***Realization*** *⌋]*

3.1.1 Descriptors

The descriptor of a reusable business component gives an answer to the following question: "when and why use this component?". A descriptor has an *intention* and a *context*. The intention is the expression of the generic modeling problem; the term "generic" here means that this problem does not refer to the context in which it is supposed to be solved. The context of a reusable business component is the knowledge which explains the choice of one alternative and not the other. Formally, descriptors are defined by the following schemas:

Descriptor = = *[intention :* ***Intention*** *;*
context : ***Context*** *⌋]*

Intention = = *[action:* ***EnginneeringActivity****;*
target: ***Interest*** *⌋]*

Context = = *[domain :****Domain*** *;*
process : $\mathbb{F}$ ***Context*** *⌋]*

EngineeringActivity = = ***AnalysisActivity*** / ***DesignActivity***

AnalysisActivity = *{analyze, ...}*

DesignActivity = *{design, decompose, describe, specify ,...}*

The detailed specification is given in [1]. For the intelligibility of this paper, we give below an important type used in the above specification: *Interest*.

The engineering activity defined in the intention (hereafter referred to as the action of the intention) of a reusable business component acts on a "target" which can be a business domain or a set of business objects. Here are two examples of intentions formalized in FORM/BCS :

- *(analyze)*$_{ACTION}$*(civil servant management system)*$_{TARGET}$
- *(describe)*$_{ACTION}$*(civil servant recruitment application)*$_{TARGET}$

Interests of engineering activities are specified by the following schemas in which $\mathbb{F}A$ denotes the set of finite subsets of A.

Interest = ***Domain*** ⎪ ***BusinessObjects***

Domain = = *[action:* ***BusinessActivity****;*
target : ***BusinessObjects*** *;*
precision : ***Precision*** *⎪]*

BusinessObjects = = $\mathbb{F}$ ***Class***

Class = = *[name:* ***Name****;*
attributs : $\mathbb{F}$ ***Attribut****;*
operations : $\mathbb{F}$ ***BusinessActivity*** *⎪]*

Precision

Name

Attribut

A business activity is a set of (sub) business activities divided into three disjoint categories: the set of common (sub) business activities of the activity which indicate reuse opportunity (the commonality of the business activity), the set of optional (sub) business activities of the activity (the options of the business activity) and, the set of groups of alternate (sub) business activities of the activity (the variability of the business activity).

BusinessActivity == *[common:* $\mathbb{F}$
BusinessActivity *;*
optional: $\mathbb{F}$ ***BusinessActivity****;*
variabilities: $\mathbb{F}$ $\mathbb{F}$ ***BusinessActivity*** *⎪]*

3.1.2 Realizations

The realization section of a reusable component provides a *solution* to the modeling problem expressed in the descriptor section of the component. It is a conceptual diagram or a fragment of an engineering method expressed in the form of a system decomposition, an activity organization or an object description. The goals, the activities and the objects figuring in the realization section concern the application field (product fragment) or the engineering process (process fragment).
The solution, which is the reusable part of the component, provides a product or a process fragment. The types of solutions depend on the type of reusable business component i.e a solution of a feature business component (respectively a reference business component) is a feature (respectively a reference business architecture). This solution may have *adaptation points* with values fixed at the reuse moment. Adaptation points enable the introduction of parameters in the solutions provided by reusable components. Those parameters are values or domains of values of elements of the solution.

Realization = = *[solution:* ***Solution*** *;*
adaptationpoints :
AdaptationPoints *⎪]*

Solution = = ***Feature*** ⎪
SubsystemArchitecture ⎪
ProcessArchitecture ⎪
Module

AdapatationPoints = = ***FeatureAdaptationPoints*** ⎪
SubsystemArchitecture AdaptationPoints ⎪
ProcessArchitecture AdaptationPoints ⎪
Module AdaptationPoints

Solutions and adaptation points of the different types of reusable components (feature business components, subsystem architecture business components, process architecture business components, module business components) are defined in the corresponding subsections below.

3.2 Feature business components

In FORM, a feature model of a domain gives the "intention" of that domain in terms of generic features which literally marks a distinct service, operation or function visible by users and application developers of the domain. FORM/BCS specifies a feature model of a domain as a business reusable component of that domain which captures the commonalities and differences of applications in that domain in terms of features. Feature business components are used to support both the engineering of reusable domain artifacts and the development of applications using domain artifacts.

FeatureBusinessComponent = = *[name :****Name*** *;*
*descriptor:****Descriptor****;*
realization: ***Realization*** ⎪
∀ fbc:FeatureBusinessComponent,
(solution(realization(fbc)) ∈ ***Feature*** ∧
Adaptationpoints(realization(fbc)) ∈ $\mathbb{F}$ *(****Feature*** × $\mathbb{F}$ ***Feature)]***

Feature = =*[activity:* ***BusinessActivity*** *;*
objects: ***BusinessObjectst*** *;*

decomposition:[common: $\mathbb{F}$ ***Feature****; optional:* $\mathbb{F}$ ***Feature****; variabilities:* $\mathbb{F}\mathbb{F}$***Feature]***

generalization: $\mathbb{F}$ ***Feature*** *⎪]*

In the above schemas, the type *Feature* specifies business activities. A business activity is caused by

an event which is applied to a target set of objects. Features have a generalization (in the sense of object-oriented analysis) and decomposition. A feature's decomposition gives the set of its common (sub) features which indicate reuse opportunity, the set of its optional (sub) features and the set of its groups of alternate (sub) features.

A reusable feature business component *fbc* is well formed if it satisfies the following four characteristic properties which require that the realization section of a feature business component corresponds to the intention of that business component:

(fbc1) The solution given in the realization section of *fbc* is a solution of the intended contextual business activity of *fbc*:

$$action(domain(context(descriptor(fbc)))) = activity(solution(realization(fbc)))$$

(*fbc2*) The target of the intended contextual business activity of *fbc* is exactly the set of objects collaborating in the business activity of the solution given in the realization section of *fbc*:

$$target(domain(context(descriptor(fbc)))) = objects(solution(realization(fbc)))$$

(fbc3) Any requirement expressed in the form of a business process in the intended contextual business activity of *fbc* has a unique solution in the realization section of *fbc*:

$$\forall p \in process(context(descriptor(fbc))),\ \exists !\, g \in decomposition(solution(realization(fbc))) \bullet activity(g) = action(domain(p)) \wedge objects(g) = target(domain(p))$$

(fbc4) Any solution in the decomposition of the solution of the realization of *fbc* expressed in the form of a feature resolves a unique requirement expressed in the form of a business process in the intended contextual business activity of *fbc*:

$$\forall g \in decomposition(solution(realization(fbc))),\ \exists !\, p \in process(context(descriptor(fbc))) \bullet activity(g) = action(domain(p)) \wedge objects(g) = target(domain(p))$$

3.3 Subsystem architecture business components

A subsystem architecture business component is a reusable business component which describes a system in terms of abstract high level subsystems and the relationships between them.

$$\textbf{SubSystemBusinessComponent} = = [name: \textbf{Name};\ descriptor: \textbf{Descriptor};\ realization: \textbf{Realization}/\ \forall ssbc: SubSystemBusinessComponent,\ (solution(realization(ssbc)) \in \textbf{SubsystemArchitecture} \wedge adapationpoints(realization(ssbc)) \in \mathbb{F}(\textbf{SubSystem} \times \mathbb{F}\,\textbf{SubSystem})]$$

$$\textbf{SubsystemArchitecture} = = [subsystems: \mathbb{F}\,\textbf{SubSystem};\ Links: \mathbb{F}(\textbf{Subsystem} \times \textbf{SubSystem})/]$$

$$\textbf{SubSystem} = \mathbb{F}\,\textbf{Feature}$$

A reusable subsystem business component *sbc* is well formed if it satisfies the following two characteristic properties which require that the realization section of a subsystem business component corresponds to the intention of that business component:

(sbc1) Any requirement expressed in the form of a business process in the intended contextual business activity of *sbc* has a unique solution in the realization section of *sbc*:

$$\forall p \in process(context(descriptor(sbc))),\ \exists !\, ss \in subsystems(solution(realization(sbc))),\ \exists !\, f \in ss, \bullet\ activity(f) = action(domain(p)) \wedge objects(f) = target(domain(p))$$

(sbc2) Any link in the solution of the realization section of *sbc* resolves requirements expressed as collaborations between business processes in the intended contextual business activity of *sbc*:

$$\forall (ss1, ss2) \in links(solution(realization(sbc))),\ \exists (f1, f2) \in ss1 \times ss2 \bullet decomposition(f1) \cap decomposition(f2) \neq \varnothing$$

Graphically, the solution of a subsystem architecture business component is represented as a symmetric boolean matrix in which rows and columns represent the different subsystems of the business component and the values of the matrix indicate the existence of links between these subsystems.

3.4 Process architecture business components

A process architecture business component is a reusable business component which represents a concurrency structure in terms of concurrent business activities to which functional elements are allocated; the deployment architecture shows an allocation of business activities to resources.

$$\textbf{ProcessBusinessComponent} = = [name: \textbf{Name};\ descriptor: \textbf{Descriptor};\ realization: \textbf{Realization}/\ \forall pbc: ProcessBusinessComponent,\ (solution(realization(pbc)) \in \textbf{ProcessArchitecture} \wedge adaptationpoints(realization(pbc)) \in \mathbb{F}(\textbf{BusinessActivity} \times \mathbb{F}\,\textbf{BusinessActivity})]$$

ProcessArchitecture *= =*
[tasks: 𝔽 ***BusinessActivity*** *;*
datas : 𝔽 ***Class****;*
dataaccess: 𝔽 [name: ***Name***
access: ***BusinessActivity*** *×* ***Class*** */]*
messages: 𝔽 [name: ***Name****;*
call: ***(BusinessActivity*** *U*
{null}) ×
(BusinessActivity *U*
{null}) /]
/]

In the above schemas, the type *ProcessArchitecture* specifies process architectures. A process architecture is a set of business activities and objects (data). The business activities operate on data and exchange messages between them (in the form of actions call) or with the environment (null). A reusable process architecture business component *pbc* is well formed if it satisfies the following three characteristic properties which require that the realization section of a process business component corresponds to the intention of that business component:

(pbc1) Any requirement expressed as a business process in the intended contextual business activity of *pbc* has a unique solution in the realization section of *pbc*:
∀ p ∈ process(context(descriptor(pbc))),
∀ a ∈ decomposition(action(domain(p)))
(∃ ! t ∈ tasks(solution(realization(pbc))) • t = a) ∧ (target(domain(p)) ⊆
datas(solution(realization(pbc)))) ∧ (((a, d) ∈
dataaccess(solution(realisation(pbc)))) ⇔ ((a ∈
decomposition(action(domain(p))) ∧
(decomposition(a) ∩ operations(d)) ≠ ∅)))
where *decomposition(a)* is written for *common(a) ∪ optional(a) ∪ (∪ variabilities(a))*

(pbc2) Messages are sent only between tasks having common actions:
∀ p ∈ process (context(descriptor(pbc))), (((t1, t2) ∈ decomposition (action(domain(p))) × decomposition (action(domain(p)))) ∧
(decomposition(t1) ∩ decomposition(t2) ≠ ∅))
⇔ ((t1, t2) ∈ messages
(solution(realisation(pbc))))

3.5 Module business components

Module business architecture components are refinements of process business architecture components. A module may be associated with a set of relevant features. Also, alternative features may be implemented as a template module or a higher level module with an interface that could hide all the different alternatives. The following schemas formally define module business components:

ModuleBusinessComponent *= = [name:* ***Name*** *;*
descriptor: ***Descriptor****;*
realization: ***Realization*** */*
∀ mbc:ModuleBusinessComponent,
(solution(realization(mbc)) ∈ ***Module*** *∧*
adaptationpoints(realization(mbc)) ∈ 𝔽
*(****Module*** *×* *𝔽* ***Module****)]*

Module *= = [pseudonym :* ***Name*** *;*
parameters: 𝔽 ***Parameter****;*
description: [task: ***BusinessActivity****;*
included: 𝔽 ***Module****;*
external: 𝔽 ***Module****]*
specification: ***PseudoCode*** */]*

Parameter
PseudoCode

In the above schemas, a module has a name, a list of parameters, a code in a pseudo language and a description which defines the task done by the module and the modules required for its execution, some of them are included in the module and some others are external. A module business component *mbc* is well formed if it satisfies the following characteristic property of module business components, that is:

(mbc1) The set of requirements expressed as a business process in the intended contextual business activity of *mbc* is a singleton and has a solution in the realization section of *mbc*.
(# process(context(descriptor(mbc)) = 1) ∧
((p ∈ process(context(descriptor(mbc))) ⇔
(action(domain(p)) =
task(description(solution(realization(mbc))))))

4. The formalization of FORM with the Business Component Semantics horizontal engineering process.

The formalization of the horizontal engineering process of FORM/BCS is done through four total functions, called *constructors*: (i) *SSA*, the subsystem architecture constructor, (ii) *PA*, the process architecture constructor and, (iii) *MA*, the module architecture constructor. Each constructor specifies an activity of the horizontal engineering process by defining how that activity produces a target asset from an input one.

4.1 The subsystem architecture constructor.

The subsystem architecture constructor *SSA* corresponds to the subsystem architecture design activity of the horizontal process. It defines how that activity produces a reusable subsystem

business architecture component from a reusable feature business component. The reusable subsystem business component architecture *SSA(fbc)* constructed by *SSA* from an input feature business component *fbc* is defined as:
SSA (fbc) = (SSA.name(fbc), SSA.descriptor(fbc), SSA.realization(fbc)) where:

(a) *SSA.name(fbc)* is a text used by the designer to name the reusable subsystem business component.

(b) *SSA.descriptor(fbc) = (SSA.intention(fbc), SSA.context(fbc))* with**:**
- *SSA.intention(fbc)* = $<(decompose)_{ACTION}(domain(descriptor(fbc))_{TARGET})>$, whose meaning is that the intention of *SSA(fbc)* is to decompose the business domain of *fbc.*
- *SSA.context(fbc) = context(descriptor(fbc))* which says that the context of *SSA (fbc)* is the same as the context of *fbc*.

(c) *SSA.realization(fbc) = (SSA.solution(fbc), SSA.adaptation_points(fbc))* where:
- *SSA.solution(fbc) = (SSA.subsystems(fbc), SSA.links(fbc))* where:
  - *SSA.subsystems(fbc)* is the partition of the solution of the realization of *fbc* defined as follows:
    (i) SSA.subsystems(fbc) $\subseteq \mathbb{F}\,\mathbb{F}$ *Feature*
    (ii) $\cup(F \in SSA.subsystems(fbc)) = decomposition(solution(realization(fbc)))$
    (iii) $\forall F1, F2 \in SSA.subsystems(fbc)$, $F1 \neq F2 \Rightarrow F1 \cap F2 = \varnothing)$
    (iv) $\forall F \in SSA.subsystems(fbc), \forall f \in Feature$, $\forall g \in Feature$, $((f \in F \wedge g \in F) \Leftrightarrow (\exists h \in F / (objects(f) \cap objects(h) \neq \varnothing) \wedge (objects(g) \cap objects(h) \neq \varnothing)))$.
  - *SSA.links(fbc)*= $\{(F,G) \in SSA.subsystems(fbc) \times SSA.subsystems(fbc) / \exists(f,g) \in F \times G \bullet decomposition(f) \cap decomposition(g) \neq \varnothing\}$
  - *SSA.adaptation_points(fbc)* = $\{(ss, subsystemrealizations(ss)) \bullet ss \in SSA.subsystems(fbc) \wedge ss \cap adaptation\_points(realization(fbc)) \neq \varnothing\}$ where *subsystemrealizations(ss)* = $\{ss':Subsystem \bullet \forall f \in ss, \exists! g \in ss' / g \in featurerealizations(f) \wedge \forall g \in ss', \exists f \in ss / g \in featurerealizations(f)\}$ and *featurerealizations(f)* = $\{g:Feature \bullet common(f) \subseteq common(g) \wedge \forall V \in variabilities(f)$**,** $(\exists ! h \in common(g) \bullet h \in V) \wedge optional(g) \subseteq optional(f)\}$

Lemma 1: if *fbc* is a well formed feature business component, then *SSA(fbc)* is a well formed subsystem business component.

Proof: It suffices to show that *SSA(fbc)* verifies the subsystem business components' characteristics defined in section 3.3:

sbc1) Suppose $p \in process(SSA.context(fbc))$, we must show that $\exists! ss \in SSA.subsystems(fbc))$ and $\exists! f \in ss$ such that $activity(f) = action(domain(p)) \wedge objects(f) = target(domain(p))$.
Since $SSA.context(fbc) = context(descriptor(fbc))$ and *fbc* is a well formed feature, then $\exists! f \in decomposition(solution(realization(fbc)))$ such that $activity(f) = action(domain(p)) \wedge objects(f) = target(domain(p))$ (property fbc3).
By definition *SSA.subsystems(fbc* is a partition of $decomposition(solution(realization(fbc)))$, hence $\exists! ss \in SSA.subsystems(fbc))$ such that $f \in ss$.

sbc2) Suppose $(ss1, ss2) \in SSA.links(fbc)$, we must show that $\exists\ (f1, f2\ ) \in ss1 \times ss2$ such that $decomposition(f1) \cap decomposition(f2) \neq \varnothing$.
This property is trivially true by the definition of *ssa.links*.

Proposition 1: Let *fbc* be a well formed feature business component, all the requirements expressed in *fbc* have a solution in *SSA(fbc)* ie:

i) If $p \in process(context(fbc))$ then $\exists ss \in SSA.subsystems(fbc)$ such that $\exists f \in ss$ and $activity(f) = action(domain(p))$

ii) If $f \in decomposition(solution(realization(fbc)))$ *and* $f \in adaptation\_points(fbc)$ then $\exists ss \in SSA.subsystems(fbc)$ such that $f \in ss$ *and* $ss \in SSA.adaptation\_points(fbc)$

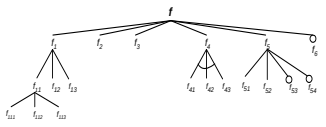
Proof: The proof of (i) is obvious since *SSA(fbc)* has the same context as *fbc* and *SSA(fbc)* is a well formed subsystem business component if *fbc* is a well formed feature business component (lemma1). The proof of (ii) is also obvious by the definition of *SSA.adaptation_points*.

Example: Let us consider the following skeleton of a feature business component, hereafter referred as FM/IMCS.

Name : Functional Model of the Integrated Management of Civil Servants and Salaries in Cameroon
Intention : $(Define)_{ACTION}((manage)_{ACTION}(civil\ servants\ and\ salaries)_{TARGET})_{TARGET}$
Context :
Domain : f = $(manage)_{ACTION}(carrers,payroll,training,network,mail,system)_{TARGET}$
Business processes :
f1 = $(manage)_{ACTION}(careers)_{TARGET}$
f2 = $(manage)_{ACTION}(payroll)_{TARGET}$
f3 = $(manage)_{ACTION}(training)_{TARGET}$
f4 = $(manage)_{ACTION}(inter\text{-}ministerial\ network)_{TARGET}$
f5 = $(administer)_{ACTION}(the\ system)_{TARGET}$
f6 = $(manage)_{ACTION}(mail)_{TARGET}$
/* sub-processes of f1 */
f11 = $(manage)_{ACTION}(recruitment)_{TARGET}$
f12 = $(manage)_{ACTION}(promotion)_{TARGET}$
f13 = $(transfer)_{ACTION}(file)_{TARGET}$
/* sub-processes of f2 */
f21 = $(transfer)_{ACTION}(file)_{TARGET}$
f22 = $(calculate)_{ACTION}(salaries)_{TARGET}$
f23 = $(manage)_{ACTION}(users)_{TARGET}$
f24 = $(manage)_{ACTION}(profiles, users)_{TARGET}$
/* *sub-processes of f4* */
f41 = $(use)_{ACTION}(optic\ fiber\ network)_{TARGET}$
f42 = $(use)_{ACTION}(radio\ network)_{TARGET}$
f43 = $(use)_{ACTION}(twisted\ pair\ network)_{TARGET}$
/* *sub-processes of f5* */
f51 = $(manage)_{ACTION}(users)_{TARGET}$
f52 = $(manage)_{ACTION}(profiles, users)_{TARGET}$
f53 = $(manage)_{ACTION}(connexions, users)_{TARGET}$
f54 = $(manage)_{ACTION}(the\ audit\ track, users)_{TARGET}$

/ sub-processes of f11 */*
f_{111} = (integrate)$_{ACTION}$ (civil servant)$_{TARGET}$ (if he has passed a competitive examination
or he has a diploma giving right to integration)$_{PRECISION}$
f_{112} = (sign recruitment order)$_{ACTION}$ (civil servant)$_{TARGET}$ (if the prime minister office has authorized)$_{PRECISION}$
etc…

Solution :



Adaptation *Points* : {(f, realizations(f)), (f4, realizations(f4)), (f5, realizations(f5))}
End.

Due to limited space, the complete specification of the components of this ongoing real case study cannot be given. Only some relevant sections are shown in order to give the intuition to the reader. A skeleton of the sub-system architecture business component SSAM/IMCS derived from FM/IMCS is given below:

Name : Subsystem architecture of the Integrated Management of Civil Servants and Salaries in Cameroon
Intention : (Decompose)$_{ACTION}$((manage)$_{ACTION}$(civil servants and salaries)$_{TARGET}$)$_{TARGET}$

…

Solution:
Subsystems:
SS1 = {f1}
SS2 = {f2}
SS3 = {f3}
SS4 = {f4}
SS5 = {f5}
SS6 = {f6}

Links: SS1 ↔ SS2 , SS2 ↔ SS5

Adaptation *Points* : {(SS4, …) , (SS5, …)}
End.

4.2 The process architecture constructor.

The process architecture constructor *PA* formalizes the process architecture design activity of the horizontal process. It defines how that activity produces a set of process business architecture components from a reusable subsystem business component. The set of reusable process business component architectures *PA(sbc)* constructed by *PA* from an input subsystem business component *sbc* is defined as follows:

PA(sbc) = {(PA.name(p), PA.descriptor(p), PA.realization(p)) • p ∈ process(sbc)} where

(a) *PA.name (p)* is a text used by the designer to designate the process business component.

(b) *PA.descriptor(p) = (PA.intention(p), PA.context(p))* where

- *PA.intention(p) = (describe)$_{ACTION}$ (p)$_{TARGET}$* whose meaning is that the intention of the process architecture built from *p ∈ process(sbc)* is to describe *p.*
- *PA.context(p) = (domain(sbc), {p})* This says that the domain of the context of the process architecture constructed from the process *p ∈ process (sbc)* is the same as the domain of *sbc.* The set of his processes has a single process equals to *p*..

(c) *PA.realization(p) = (PA.solution(p), PA.adaptation_points(p))* where

- *PA.solution(p) = (PA.tasks(p), PA.datas(p), PA.dataaccess(p), PA.messages(p))* where
  PA.tasks(p) = decomposition(action(domain(p)));
  That is tasks of the process architecture of a process *p* in *process(sbc)* are tasks obtained by decomposing *action(domain(p))*.
  PA.datas(p) = target(domain(p));
  whose meaning is that the set of the process architecture of a process *p* in *process(sbc)* corresponds to the target of the domain of *p*
  PA.dataaccess(p) = {(t, c) ∈ decomposition(action(domain(p))) × target(domain(p)) / decomposition(t) ∩ operations(c) ≠∅}
  That is data access of the process architecture of a process *p* in *process(sbc)* are pairs composed with tasks obtained by decomposing p and class of *target(p).*
  PA.messages(p) = {(t1, t2) ∈ (decomposition (action(domain(p))))^2 / decomposition (t1) ∩ decomposition (t2) ≠∅}.
  This means that messages of the process architecture of a process *p* in *process(sbc)* are pairs of tasks *(t1, t2)* obtained by decomposing *action(domain(p))* such that *decomposition (t1) ∩ decomposition (t2) ≠∅.*
- *PA.adaptation_points(p)) = {(t1, A) • t1 ∈ PA.tasks(p) ∧*
  A = {t2 : BusinessActivity • common(t1) ⊆ common(t2) ∧
  (∀ V ∈ variabilities(t1), ∃ ! g ∈ common(t2) • g ∈ V) ∧
  optional(t1) ⊆ optional(t2)} ∧ #A > 1}.
  That is, adaptation points of the process architecture of a process *p* in *process(sbc)* are tasks of the process architecture of *p* for which we have more than one realization.

Lemma 2: if *sbc* is a well formed subsystem business component, then *(PA.name(p), PA.descriptor(p), PA.realization(p))* such that *p ∈ process(sbc)* is a well formed process business component.

Proof: It is sufficient to show that *(PA.name(p), PA.descriptor(p), PA.realization(p))* verifies the process business components' characteristics defined in section 3.4:

pbc1) Suppose $q \in process(PA.context(p))$, $a \in decomposition(action(domain(q)))$ we must show that
$\exists! t \in PA.tasks(p) \bullet t = a \wedge target(domain(q)) \subseteq PA.datas(p) \wedge ((((a, d) \in PA.dataaccess(p) \Leftrightarrow ((a \in decomposition(action(domain(q)))) \wedge ((decomposition(a) \cap operations(d) \neq \varnothing)))$
where *decomposition(a)* is written for $common(a) \cup optional(a) \cup (\cup variabilities(a))$.
$q \in process(PA.context(p)) \Rightarrow q = p$ (1)
basing on (1) $a \in decomposition(action(domain(q))) \Rightarrow a \in decomposition(action(domain(p)))$.
By the definition of *PA.tasks*, $PA.tasks(p) = decomposition(action(domain(p)))$, hence $\exists! t \in PA.tasks(p) \bullet t = a$
According to the definition of *PA.datas*, $PA.datas(p) = target(domain(p))$, and basing on (1) we conclude that $target(domain(q)) \subseteq PA.datas(p)$
By the definition of *PA.dataaccess*, $((((a, d) \in PA.dataaccess(p) \Leftrightarrow ((a \in decomposition(action(domain(q)))) \wedge ((decomposition(a) \cap operations(d) \neq \varnothing)))$

pbc2) Suppose $q \in process(PA.context(p))$, we must show that $((t1, t2) \in decomposition(action(domain(q))) \times decomposition(action(domain(q))) \wedge (decomposition(t1) \cap decomposition(t2) \neq \varnothing)) \Leftrightarrow ((t1, t2) \in PA.messages(p))$
$q \in process(PA.context(p)) \Rightarrow q = p$ (1)
By the definition of *PA.messages*, $PA.messages(p) = \{(t1, t2) \in (decomposition(action(domain(p))))^2 / decomposition(t1) \cap decomposition(t2) \neq \varnothing\}$ (2)
Basing on (1) and (2), we conclude that $((t1, t2) \in decomposition(action(domain(q))) \times decomposition(action(domain(q))) \wedge (decomposition(t1) \cap decomposition(t2) \neq \varnothing)) \Leftrightarrow (t1, t2) \in PA.messages(p)$

Proposition 2: Let *sbc* be a well formed subsystem business component, all the requirements expressed in *sbc* have a solution in *PA(sbc)* ie:

i) If $p \in process(sbc)$ then $\exists pa \in PA(sbc)$ such that
$pa = (PA.name(p), PA.descriptor(p), PA.realization(p))$

ii) If $ss \in subsystems(solution(realization(sbc)))$, $f \in ss$ and $g \in decomposition(f)$ such that $decomposition(g) \neq common(g)$ then $\exists t \in PA.tasks(p)$ such that $action(domain(p)) = activity(f)$, $t = activity(g)$ and $t \in PA.adaptation\_points(p)$ where $p \in process(sbc)$.

Proof: The proof of (i) is obvious since $PA(sbc) = \{(PA.name(p), PA.descriptor(p), PA.realization(p)) \bullet p \in process(sbc)\}$. The proof of (ii) is also obvious by the definition of *PA.adaptation_points*.

Example:
By applying the previous passage rule to the process f1 = (manage = [{recruit, promote, liquidate, transfer}, {}, {}]) ACTION (candidates, requests, competitive examinations, civil servants, deeds) TARGET of the previous subsystem business component, we obtain the process business component below named PAM/RM:

Name: Process architecture model of careers management.
Intention : (describe)$_{ACTION}$((manage = [{recruit, promote, liquidate, transfer}, {}, {}])$_{ACTION}$(candidates, requests, competitive examinations, civil servants, deeds)$_{TARGET}$)$_{TARGET}$
Context :
Domain : f = (manage)$_{ACTION}$(careers,payroll,training,network,mail,system)$_{TARGET}$
Process : f1 = (manage = [{recruit, promote, liquidate, transfer}, {}, {}])$_{ACTION}$(candidates, requests, competitive examinations, civil servants, deeds)$_{TARGET}$
/* sub process of the process f1 */
f11 = (recruit = [{initiate, validate, visa, sign}, {modify, delete, remove validation}, {}])$_{ACTION}$(candidates, requests, competitive examinations, civil servants, deeds)$_{TARGET}$(If he has succeeded to a competitive examination or he has a diploma giving right to integration or the presidency of the republic has gave agreement)$_{PRECISION}$
f12 = (promote = [{initiate, validate, visa, sign}, {modify, delete, remove validation}, {}]) $_{ACTION}$ (requests, civil servants, deeds) $_{TARGET}$
f13 = (liquidate = [{initiate, validate, visa, sign}, {modify, delete, remove validation}, {}]) $_{ACTION}$ (requests, civil servants, deeds) $_{TARGET}$
f14 = (transfer = [{initiate, validate, visa, sign}, {modify, delete, remove validation}, {}]) $_{ACTION}$ (civil servants, deeds) $_{TARGET}$
…
Solution **:**
tasks: decomposition(action(f1))
datas: target(f1)
dataaccess: {(t, c) ∈ decomposition(action(f1)) × target(f1) / decomposition(t) ∩ operations(c) ≠∅}
messages: {(t1, t2) ∈ decomposition (action(f1)) × decomposition (action(f1)) / decomposition (t1) ∩ decomposition (t2) ≠∅}
Adaptation points **:**
$\{(t1, A) \bullet t1 \in pa.tasks(f1) \wedge A = \{t2 : BusinessActivity \bullet common(t1) \subseteq common(t2) \wedge (\forall V \in variabilities(t1), \exists ! g \in common(t2) \bullet g \in V) \wedge optional(t1) \subseteq optional(t2)\} \wedge \#A > 1\}$
End.

4.3 The module architecture constructor.

The module architecture constructor *MA* corresponds to the module architecture design activity of the horizontal process. It defines how that activity produces a set of module business architecture components from a reusable process business component. The set of reusable module business component architectures *MA(pbc)* constructed by *MA* from an input process business component *pbc* is defined as:

MA(pbc) = {(MA.name(t), MA.descriptor(t), MA.realization(t)) • t ∈ process(p), p ∈ process(pbc)} where *:*

(a) *MA.name (t)* is a text used by the designer to name the module business component.

(b) *MA.descriptor(t) = (MA.intention(t), MA.context(t))* where

- *Ma.intention(t) = <(specify)$_{ACTION}$ (t)$_{TARGET}$>* meaning that the intention of the module architecture built from *t* is to specify *t.*
- *MA.context(t) = (domain(pbc), {t})* This says that the domain of the context of the module architecture constructed from the task *t is* the same as the domain of *pbc.* The set of these processes has a single process equals to *t*..

(c) *MA.realization(t) = (MA.solution(t), MA.adaptation_points(t))* where

- *MA.solution(p) = (MA. pseudonym (t), MA.parameters(t), MA.task(t), MA.included(t), MA.external(t), MA.specification(t))* where
  MA. pseudonym (t) is a text used by the designer to name the solution;
  MA.parameters(t) is a set of texts pointing out business objects (or classes) in the target of the domain of *t;*
  Ma.task(t) = action(domain(t))
  That is the task of the solution of the module architecture constructed from the context *t* is the action of the domain of *t.*
  MA.included(t) = {m:Module • task(m) ∈ decomposition(action (domain(t))) ∧ specification(m) ≠ "" };
  Meaning that, modules included in the module architecture constructed from the context *t* are module for which the task is a subtask of *t* and the specification is not empty.
  MA.external(t) = {m:Module • task(m) ∈ decomposition(action (domain(t))) ∧ specification(m) = "" };
  Meaning that, external modules in the module architecture constructed from the context *t* are module for which the task is a subtask of *t* and the specification is empty.
  MA.specification(t) = {specification(m) • m ∈ MA.included(t) ∪ MA.required(t)}.
  That is the specification of the module architecture constructed from the context *t* is the set of specifications of subtasks of *t.*
- *MA.adaptation_points(t)) = {(m1, A) • m1 ∈ MA.included(t) ∧ A = {m2 : Module • common(task(m1)) ⊆ common(task(m2)) ∧ (∀V ∈ variabilities(task(m1)), ∃ ! g ∈ common(task(m2)) • g ∈ V) ∧ optional(task(m1)) ⊆ optional(task(2))} ∧ #A > 1}*. That is, adaptation points of the module architecture of a context *t* are modules included in the module architecture of *t* for which we have more than one realization.

Lemma 3: if *pbc* is a well formed process business component, then *(MA.name(t), MA.descriptor(t), MA.realization(t))* such that *t ∈ process(p)* and *p ∈ process(pbc)* is a well formed module business component.

Proof: It suffices to show that *(MA.name(t), MA.descriptor(t), MA.realization(t))* verifies the module business components' characteristics defined in section 3.5:

mbc1) Suppose *p ∈ process(pbc)* and *t ∈ process(p),* we must show that *(#MA.process(t) = 1) ∧ ((p ∈ MA.process(t)) ⇔ (action(domain(p)) = MA.task(t))*.
This property is trivially true by the definition of *MA.process*.

Proposition 3: Let *pbc* be a well formed process business component, all the requirements expressed in *pbc* have a solution in *MA(pbc)* ie:

i) If *p ∈ process(pbc)* and *t ∈ process(p)* then *∃ ma ∈ MA(pbc)* such that *ma = (MA.name(t), MA.descriptor(t), MA.realization(t))*

ii) If *t ∈ tasks(solution(realization(pbc)))* and *a ∈ decomposition(t)* such that *decomposition(a) ≠ common(a)* then *∃ m ∈ MA.adaptation_points(t)* such that *task(m) = a.*

Proof: The proof of (i) is obvious since *MA(pbc) = {(MA.name(t), MA.descriptor(t), MA.realization(t)) • t ∈ process(p), p ∈ process(pbc)}*. The proof of (ii) is also obvious by the definition of *MA.adaptation_points*.

Example:

By applying the previous passage rule to the task *integrate* of the previous process business component, we obtain the module business component below named MAM/ICS:

Name: Module business architecture of the recruitment of a civil servant.

Intention : (specify) $_{ACTION}$ ((recruit = [{initiate, validate, visa, sign}, {modify, delete, remove validation}, {}])$_{ACTION}$(candidates, requests, competitive examinations, civil servants, deeds)$_{TARGET}$(If he has succeeded to a competitive examination or he has a diploma giving right to integration or the presidency of the republic has gave agreement)$_{PRECISION}$) $_{TARGET}$

Context :

Domain : f = (manage)$_{ACTION}$(careers,payroll,training,network,mail,system)$_{TARGET}$

Process: f11 = (recruit = [{initiate, validate, visa, sign}, {modify, delete, remove validation}, {}])$_{ACTION}$(candidates, requests, competitive examinations, civil servants, deeds)$_{TARGET}$(If he has succeeded to a competitive examination or he has a diploma giving right to integration or the presidency of the republic has gave

agreement)$_{PRECISION}$
/* sub process of the process f11 */
(initiate)$_{ACTION}$ ({Deed, Servant })$_{TARGET}$
(validate)$_{ACTION}$ ({Deed})$_{TARGET}$
(visa)$_{ACTION}$ ({Deed})$_{TARGET}$
(sign)$_{ACTION}$ ({Deed})$_{TARGET}$
(modify)$_{ACTION}$ ({Deed})$_{TARGET}$
(delete)$_{ACTION}$ ({Deed})$_{TARGET}$
(removevalidation)$_{ACTION}$ ({Deed})$_{TARGET}$

Solution :

pseudonym : recruit;
parameters: {candidates, requests, competitive examinations, civil servants, deeds};
task: <{initiate, validate, visa, sign}, {modify, delete, remove validation}, {}>;
included: *{m:Module • task(m) ∈ decomposition(action (domain(recruit))) ∧ specification(m) ≠ ""}*;
external: *{m:Module • task(m) ∈ decomposition(action (domain(recruit))) ∧ specification(m) = ""}*;
specification: *{specification(m) • m ∈ Ma.included(recruit) ∪ Ma.required(recruit)}*

Adaptation Points :

{(m1, A) • m1 ∈ Ma.included(recruit) ∧
A = {m2 : Module • common(task(m1)) ⊆ common(task(m2)) ∧
(∀ V ∈ variabilities(task(m1)), ∃ !g ∈ common(task(m2)) • g ∈ V)
∧ optional(task(m1)) ⊆ optional(task(m2))} ∧ #A > 1}

End.

5. Related works.

The scientific community has a lot of interest for feature-oriented approaches in product line engineering. A recent and exhaustive overview of feature-oriented development done by Sven Apel and Christian Kästner [15] points to connections between different lines of research and identifies open issues following the phases of the feature-oriented development process: domain analysis, domain design and specification, domain implementation, product configuration and generation, feature-oriented development theory.

The issues addressed in this paper mainly concern domain analysis and domain design and specification. Concerning domain analysis, according to Sven Apel and Christian Kästner, the main challenge is to reconcile the two field uses of feature models, which are useful for the communication between stakeholders on the one hand and the automation of the development process on the other hand. Concerning this issue, the formalism used in this work has a simple and intuitive syntax which enables modeling of domains in a natural way. Nevertheless, the Z notation which gives formal semantics to our business feature models provides a framework for a rigorous analysis of the method and opens the door, through the given mapping rules, for a possible automation of the development process.

Another major line of research is aimed at enriching feature models with additional information which can be used to guide the configuration and generation process. But, the more information is exposed to model, the more complex the model becomes. To avoid this complexity, additional information are not added to the feature model but, a new model called *feature business component model* is proposed. This new model has two parts: the solution part which can be any classical feature model and the contextual part whose aim is to increase the understanding of the solution part, during the product configuration and generation process in order to rule out invalid or suboptimal product variants.

Concerning the domain design and specification, that is the product line architecture definition process, remarkably, there has not been much work. Researchers concentrated mainly on feature modeling and feature implementation. This is in stark contrast to non- feature-oriented approaches, in which developers have to design product line architecture first in order to define the granularity of components or extension points in a common framework. This work gives a set of mapping rules which build product line architecture from its feature model as in [16]

Concerning domain implementation, one key issue is to establish a one-to-one mapping between features that appear during the domain analysis and feature that appear at the implementation level. Although the Domain implementation is not considered in this work, its importance is taken into consideration since the main theorem of the work given below, trivially proven from propositions 1, 2, and 3, shows that all the features that appear during the domain analysis (i. e. in the feature model) have a solution in the derived product line architecture.

Theorem: All requirements expressed in a well formed feature business component *fbc* have a solution in *MA (pbc)* where *pbc ∈ PA(SSA(fbc)))*, that is:

i) If *p ∈ process(context(fbc))* and *t ∈ process(p)* then *∃ ma ∈ MA(pbc)* with *pbc ∈ PA (SSA (fbc)))* such that *ma = (MA.name(t), MA.descriptor(t), MA.realization(t))*.

ii) If *t ∈ tasks(solution(realization(pbc)))* with *pbc ∈ PA (SSA (fbc)))* , and *a ∈ decomposition(t)* such that *decomposition(a) ≠ common(a)* then *∃ m ∈ MA. adaptation_points (t)* such that *task(m) = a.*

6. Conclusion

The goal of this article was to formalize the FORM/BCS horizontal engineering process. We have shown that this process can be systematized through a set of maps describing how one can systematically and rigorously derive the fundamental business architectures of a product line from the feature model of that domain. This derivation is currently done first at a high abstraction level and the business architectures obtained have to be refined to integrate more operational details. Therefore, our next

investigation field concerns the formalization of FORM/BCS vertical engineering process which deals with the refinement of business process components.

References

[1] M. Fouda, Amougou Ngoumou, "The Feature Oriented Reuse Method with Business Component Semantics", *International Journal of Computer Science and Applications*, Vol. 6, No. 4, 2009, pp 63-83.

[2] K. C. Kang, S. Kim, E. Shin, and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", *Annals of Software Engineering*, Vol. 5, 1998, pp. 143-168.

[3] K.C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, Vol. 19, no. 4, July/Aug. 2002, pp. 58-65.

[4] K. Lee, K. C. Kang, W. Choi, "Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse", *Software-Practice and Experience*, 30, 2000, pp.1025-1046.

[5] K.C. Kang et al., "Feature-Oriented Product Line Software Engineering: Principles and Guidelines," *Domain Oriented Systems Development: Perspectives and Practices*, K. Itoh et al., eds., 2003, pp. 29-46.

[6] V. Pujalte, P. Ramadour, C. Cauvet, "Recherche de composants réutilisables : une approche centrée sur l'assistance à l'utilisateur", in *Actes du 22ème congrès Inforsid, Biarritz, , 25-28 may 2004*, pp. 211-227.

[7] P. Ramadour, C. Cauvet, "Approach and Model for Business Components Specification", Proceeding of the 13th International Conference on Database and Expert Systems Applications, *Lecture Notes In Computer Science*; Vol. 2453, 2002, pp 628-637.

[8] F. Barbier, C. Cauvet, M. Oussalah, D. Rieu, S. Bennasri, C. Souveyet, "Composants dans l'Ingénierie des Systèmes d'Information: Concepts clés et techniques de réutilisation ", in *Assises du GDR I3, Nancy*, Décembre 2002.

[9] W. B. Frakes, K. Kang, "Software Reuse Research : Status and Future", *IEEE Transactions on Software Engineering*, Vol.. 31, N°.7, July 2005.

[10] D.M. Weiss and C.T. R. Lai, "Software Product-Line Engineering: A Family-Based Software Development Process". Addison-Wesley, 1999.

[11] W. Frakes, R. Prieto-Diaz, and C. Fox, "DARE: Domain Analysis and Reuse Environment", *Annals of Software Engineering*. vol. 5, 1998, pp. 125-141.

[12] C. Atkinson et al., "Component-Based Product Line Engineering with UML". Addison-Wesley, 2002.

[13] R. Ommering et al., "The Koala Component Model for Consumer Electronics Software," *Computer*, vol. 33, no. 3, 2000, pp. 78-85.

[14] Y. Jia, Y. Gu, "The Representation of Component Semantics: A Feature-Oriented Approach", in *Proceedings of the 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems*, Lund University, Lund, Sweden, April 8-11, 2002.

[15] Sven Apel, Christian Kästner, "An Overview of Feature-Oriented Software Development", *Journal of Object Technology*, Vol. 8, no. 5, July-August 2009, pp. 49-84.

[16] C. Zhu, Y. Lee, W. Zhao, J. Zhang, "A feature oriented approach to mapping from domain requirements to product line architecture", Proceeding of 2006 International Conference of Software Engineering Research & Practice (SERP'06/ISBN#:1-932415-92-0/CSREA), Editor: Hamid R. Arabnia and Hassan Reza, pp 219-225, Las Vegas, USA, 2006.

Marcel Fouda Ndjodo received his MSc (1988), PhD (1992) in Mathematics and Computer science from Université of Aix-Marseille II (France). He is currently a Senior Lecturer and Head of Department of Computer Science and Instructional Technology at the Higher Teachers' Training College, University of Yaoundé I (Cameroon). His research interests include Formal Methods, Workflow Modeling, Domain Engineering and Computer technology applied to education.

Amougou Ngoumou received his MSc (2001) in Computer science from the Faculty of Science of the University of Yaoundé I (Cameroon). He is currently a Lecturer at the Institute of technology of the University of Douala and a PhD student at the Faculty of Science of the University of Yaoundé I. His main research interests include Feature-oriented Domain Engineering and Component-Based Software Development.