A Model based Business Process Requirement Rule Specification

Atsa Etoundi Roger
Department of Computer Sciences,
University of Yaoundé I,
Yaoundé, Cameroon

Fouda Ndjodo Marcel
Department of Computer Sciences,
University of Yaoundé I,
Yaoundé, Cameroon

Atouba Christian Lopez
Department of Computer Sciences,
University of Yaoundé I,
Yaoundé, Cameroon

ABSTRACT
The management of requirement is as complex as there are misunderstanding between designers and end users. To tackle this complexity, we present a business rules specification framework. This approach is defined in an incremental manner. The formalism to specify business rules is first defined for the definition of the basic model, this model is then extended by various concepts for a better comprehension of requirements by both designers and users. This concept includes the notion of equivalence between business rules, refinement of business rules and business rules constraints. Having defined the model, its impact on requirement identity, sub-requirement and requirement sub division are presented.

General Terms
Requirement Modeling

Keywords
Business Process Modeling, Requirement Engineering, Software Component, Specification language, Application Engineering, Requirement representation, Business Rule Specification, Business Rule Formalization.

1. INTRODUCTION
In [1] a goal oriented approach was defined, for the development of a business process requirements model. A formalism for the identification of user requirements was proposed. With these identified requirements, and as from the work of [4], a formal representation of the latter was defined. This representation enabled us to build a hierarchical business process requirement model. It was shown, as from [2, 3] that, this representation allowed to comprehensively describe a business process on the one hand, and on the other, that the set of expressed requirement is a set arranged hierarchically. The objective of the works [1, 2, 3], ultimately, is to develop a component-based development framework from a requirement specification very close to human language. To achieve this objective, it was initiated in [1] the definition of a goal-oriented hierarchical requirement model associated with business processes. In this model, two concepts are important and interrelated; the goal defines a potential expectation of a user and the business rule describes how to achieve that goal. In this context, it becomes indispensable for business rules, to support the various concepts developed in [1] and applicable to goals. Notably, equivalence (identity), sub division, refinement, and extension. We believe that once these concepts are verified, the requirements specification approach shown in [1], will minimize misunderstandings discussed in [5, 6, 7], between computer applications designers and users of such applications. It is for this reason that we pointed out in [1] that serious investigations should be done on the formalized specification of business rules. We hoped that this semantics description formalism (service rendered) of a business rule, shall permit us on the one hand to extend the concepts of equivalence sub-division, refinement, and extension of business rules; and on the other, to build the set of objects S^b (with b any knowledge bit) of the organization on which the business rule operates.

In literature, a number of approaches to describing business rules exist. The most commonly referenced are: SBVR (Semantics of Business Vocabulary and Business Rules) [9, 10], [11], OCL (Object Constraint Language) [18], ERML (e-Citiz Rule Markup Language) [8], PRR (Production Rule Representation) [13]. However, these are, in most cases more oriented toward expert systems. Moreover, they do not integrate the possibility to apply concepts mentioned on business rules in operations. Notably: comparison, composition, sub-division, refinement, extension. We believe that these operations are extremely important insofar as they are very often, orienting in the case of component-based software Engineering, designers of computer applications on identification, and selection of reusable aspects of systems. In addition, depending on the nature of the operations mentioned above, business rules affect the types of relationships defined between distinct components. We firmly believe that, these business rules, when "well" specified, permit us to exhaustively describe the various objects of the organization brought into play during the course of a process or a task controlled by that business rule. We firmly believe that in the case of the composition of components, software architects implicitly compose the rules embedded in these components. With this opinion, and as aforesaid, approaches of business rules specification do not seem to dwell either on the deduction of objects S^b of the organization at stake in a business rule b, nor on the need to throw more light on the concepts of equivalence, difference, extensions, sub-division or refinement of business rules; concepts dear to the approach developed in [1]. We believe that when a business rule is clearly defined, it is rich enough to inform us about the nature of the organizations' objects involved and the relationships between them. Thus, it becomes judicious to define a business rules specification approach that takes into account the concerns raised above. We hope that if business rules are properly specified, we can deduce all the objects of the organization involved in a business process, their different properties or attributes, as well as existing relationships between these objects. We believe that this work will contribute significantly in the model driven development, insofar as it provides a formalized specification of the semantics of a business activity.

In the following, we shall in Section 2, present the business rule concept; in Section 3, present the specification of a rule in our context; in Section 4, formally define the various operations and concepts applicable to business rules; in Section 5; we shall finish with the conclusion and future works.

2. "BUSINESS RULE" CONCEPT

The "business rule" concept has widely been developed in the context of expert systems. Known by the acronym BRMS ("Business Rules Management Systems"), tools for business rules management are derived from expert systems, which had their heyday in the 1980s [11]. They use similar wordings and all have at the center, an inference engine. Under the business rules-oriented approaches (BROA), this inference engine, is called business rules manager [8]. It aims at modeling human reasoning, behavior and cognitive mechanisms of a human expert in a particular domain. For this, it relies on a fact base (working memory), an inference engine (rules engine) and a base of rules (production memory) [8]. Facts concepts, inference engine will not be developed as part of this work.

2.1 Business Rule

In literature, the concept of "business rule" was widely used, each author with his own vision. Thus,

- [15, 16] support that a business rule is a formulation that defines or constrains some aspect of a business activity. Its purpose is to structure a business activity (policy, know-how), to control or influence the conduct of a business activity;
- meanwhile, [17] says that a "business rule" is a directive, which is meant to influence or guide the conduct of a business activity, in the aim of implementing a business policy that is formulated in response to an opportunity or a risk;
- [4] also thinks that a business rule defines a law of the domain to which the goal must conform to. It helps to organize a management process to achieve the goal. The goal, in turn, defines a potential expectation that the system can satisfy, it expresses what the user of the system wishes to do.

We believe that to better understand the semantics of a business rule, that it is necessary to situate it in a context or specific domain.

Definition (1):
A "business rule" is a directive of a domain which controls the conduct of a business activity of that domain. Its goal is to structure a business activity (policy, know-how) to control or influence the conduct of a business activity of the domain in question, in view of achieving an expected result.

[8] tells us that business rules are divided into two broad categories:

- Structural rules (indicating a necessity): they are rules that addresses how a business activity is organized or structured, the elements comprising it. Structural rules are definition complements;
- Operative rules (indicating an obligation): these are rules that control the manner in which the business activity is carried out. Unlike structural rules, operating rules are those that can be directly violated by business stakeholders.

In the following, we focus exclusively on business rules because they control the business activity.

2.2 Structure of Business Rules

[8] tells us that, depending on the type of business rule (declarative or procedural), the structure and semantics of that business rule vary. In this paper we focus on production rules. These rules are of the form ***"if. . . then. . . [else. . .]"*** and consist of the following elements:

- Declarations of variables: used to define the application context of the business rule;
- Salience (Expression) or "level of importance" is an expression whose evaluation returns an integer corresponding to the importance of the rule. It permits us to define an order in the execution of rules. This value can also be called business priority. In [1], we associated the level of importance to knowledge bits. It would be inappropriate to associate it explicitly to the business rule given that this association is implicit in [1]. Thus we have removed this information in the specification that we shall propose in the next section;
- "*If*" or condition, or "left-hand part", or predicate, or premise (expression): It constitutes the condition under which the rule can be applied; It consists of an expression that is evaluated as true or false. It is also called "body";
- "*Then*" (side effect expression) or Conclusion or "*right-hand side*": it is the body of the rule or "*head*". It represents the action to carry out if the condition part is evaluated as true. Under declarative rules, this part must be an expression without side effects.
- "*Else*" (optional and side effect Expression). It is only possible for procedural rules if they contain an "If" part. It represents the action to carry out if the condition part is evaluated as *false*.

Expressions are classified into two groups according to whether they influence the systems' state or not. There are *side effect* expressions and expressions without *side effects*. An expression is *without side effect* if and only if it does not change the state of the environment. Some expressions, on the contrary, during a business activity, do change the state of the environment. They are said to have *side effects*.

Although in [4, 8, 9, 10, 11, 13, 15, 16], we have a clear vision of what a business rule is, the fact remains that until now, researchers seem not interested in the fact that we can carry out a certain number of operations on these rules. The interest of defining a set of operations on business rules is to be able to:

- reuse business rules in the reengineering of a business process or components-based development.
- compose business rules within the framework of model driven engineering;
- decide on the quality of the formulation of a rule after its specification by experts (business executive of the organization and software experts);
- to open a research branch on the description of a business rule oriented business process;
- evaluate the quality of model composition.

[8] summarizes the formalized specification of a business rule r as follows:

$$R = (RuleLabel\ (VariableDeclaration)^{*}\ (conditions, actions))$$

where:

$$RuleLabel = (how?, Priority?, Visibility?, Refraction?, CreatedTime?, lastModificationDateTime?)$$

$$Conditions = (Expression \mid AndExpression \mid OrExpression \mid NegationExpression)^{+}$$

$$Action = ((Expression)^{*} \mid NegationExpression^{+})$$

In the following, we shall formalize the specification of a business rule as proposed in our approach. We continue subsequently by operations performed on business rules. The

term putting objects to the stage, refers to the use of the properties of each object in an expression.

3. PROPOSAL OF A BUSINESS RULES SPECIFICATION

In this section we will introduce the concepts necessary to describe a business rule according to our approach. These concepts are similar to those found in the OCL language [18] on the one hand and on the other hand to those used in BRMS [11]. Generally we maintain the structure of a business rule as defined in [8]. It should be noted in this section that we are not defining a programming language, but a manner of doing, for the specification of business rules, as such this allows us to perform a certain number of operations on these criteria.

3.1 Predefined Types

3.1.1 Simple Types

A simple type *ST* is given by (Dt,Nt,Rt,Stg) where:

- *Dt* denotes the date type which is used to specify attributes of objects referring to time;
- *Nt* denotes the number type referring integer values, decimal or real;
- *Rt* denotes the rule type which is used for the declaration of a business rule;
- *Stg* denotes the "String" type: referring to a given character or character strings.

A constant a of type Date, Number or String is denoted by "*a*". Consider a and b two elements of type String, we define the function $SubStr : String^2 \rightarrow Bool$ such that the following properties hold:

$$SubStr(a,b) = \begin{cases} True, & if\ a\ is\ o\ part\ of\ b \\ False, & if\ not \end{cases}$$

In the following, we will denote by $SimpleType$ the set of simple types mentioned above. We also denote by $\prod SimpleType$, the arbitrary choice of a simple type in $SimpleType$.

3.1.2 Compound Types

To these four, we add two suffixes: "aggregate", "views", and a complex type. $Something$

The $Something$ type, denoted by *STg*, refers to a physical object. Example: a career management tool, a stamped application, a registration certificate, a national identity card, purchase order, an invoice, etc.. , Whereas:

- the suffix "aggregate" preceded by the keyword "$NameSpace$", refers to a set of objects in the scope of the rule;
- the suffix "views" before the name of an object in the domain circumscribed by the suffix "aggregate" refers to the set of information (properties or attributes) of this object which shall be used by the business rule.

Objects listed using the suffix "aggregate" can be accompanied by the following entries:

- The mention "control" indicates that the object is obligatory, but no data of this object intervenes in the "description" part of the business rule. If such an element is to be absent, the data is declared invalid. For example in the case of advancement of incremental position, the effective presence is necessary, but it does not affect the actual processing of the said file;
- The mention "reference": it returns a value used to uniquely identify any object. When specifying the business rule, this value is not known. A predicate making use of "reference" has the following structure:

$$make\ reference\ to\ \ldots.[in\ \ldots]$$

The mention "artifact" indicates that the object in question is produced by the organization.

3.2 Declaration of Business Rules

The declaration of a business rule is done in two phases: the definition of its context, and the specification of the domain directives which controls the behavior of the business activity in question. The first part of the declaration of a business rule permits us to specify the usage context of each object of the organization involved in this business activity and the domains in which this business rule can still be used. These domains are listed immediately after the "keywords". Meanwhile the second part is reserved to describing the action that must be done to realize a business. This part is called description.

3.2.1 Definition of Context

The specification of any business rule begins by defining its usage context. The usage context of a business rule defines on the one hand, the semantic field of an object *ob* used in the description; and on the other hand, domains in which this business rule may be used (exploitation field of the rule). Formally, we shall define the context of a business rule as follows:

Context:

Keywords =
$Domain_1, Domain_2, \ldots, Domain_n$;
Data =
$NameSpace.aggregate\ \{ob\ \%\ STg : CA$
$[, ob\ \%\ STg : CA] *\};$
$object.views\ \{fd\ \%\ \prod ST\ [, fd\ \%\ \prod ST] *\}$

Where:-*fd* is either a reference to another object which may not be in $NameSpace$; either a simple name of an attribute or property of that object. *fd* is also called arguments of the suffix "views" or referenced attribute in the suffix "views".

-$\prod ST$ represents the type of field

- *CA = control | artifact*

The suffix "aggregate" defines the set of accessible objects when carrying out a business activity for which the rule is being defined. The suffix "views" allows for objects which have no mention "control" to explicitly define the set of information (properties or attributes) of this object, which must be used in the description part of that business rule. A single aggregate is authorized per business rule. Moreover, it is important to note that when an object has the mention "artifact", all its attributes must be defined with the mention "reference" following the syntax defined below. In the following, we shall call context objects, with no mention "control" in the suffix "aggregate". In the following, the concept set will be considered as a MultiSet.

Consider an object b in this context with no mention "control", we shall denote: $card\ (b)$, the number of properties of b listed

in the suffix « *views* » ; $NameSpace_C$, the set of objects referenced in the suffix "*aggregate*" of context C ; $card(NameSpace_C)$ represent the number of objects specified in the suffix " *aggregate*"; Sg_C the set of signature of objects of context C ; et $Keywords_C$, the set of words listed after the key word « keyword » in the context C. we denote by $Sig_C(b)$ the signature of b where $Sig_c$is defined by Sig_c: $NameSpace_C \rightarrow Sg_C$, and

$$Sig_C(b) = (\prod SimpleType \left[, \prod SimpleType\right]^*)$$

The signature of b, $Sig_C(b)$ shows the list of the attributes type defined by the suffix "views".

Given another object a of the same context, we denote $Tri(a,b)$, the function which returns the signature of a ordered in the order defined by that of b, where Tri is defined by *Tri:* ${NameSpace_C}^2 \rightarrow Sg_C$*, such that for every a and b :*

If $card(a) > card(b)$*, then*

$$Tri(a,b) = \left(Sig_C(a) \cap Sig_C(b)\right) \cup \left(Sig_C(a) - Sig_C(b)\right)$$

Else,

$$Tri(a,b) = \left(Sig_C(a) \cap Sig_C(b)\right)$$

Axiom 1: Equivalence of *Signature*

(1) We shall say that a and b have equal signatures if and only if:

$$\mathrm{Tri(a,b) = Sig_C(b).}$$

(2) a and b are said to be equivalent in their context.

Axiom 2: Refinement or Extension of Signature

(3) We shall say that the signature of a is a refinement of b if and only if:

$$Sig_C(a) \subset Sig_C(b)$$

(4) We shall also say that b is an extension of a.

Axiom 3: difference of signature

We say that the signatures of objects a and b are different if and only if they satisfy neither axiom 1, nor axiom 2.

Axiom 4: $NameSpace$ and $Contexts$:

Consider two contexts C and B,

(5) $NameSpace_C$ and $NameSpace_B$ are said to be equivalent if and only if:

1. $card(NameSpace_C) = card(NameSpace_B)$ and,
2. $\forall\, a \in NameSpace_B\ \exists b \in NameSpDce_C, Sig_B(a) = Sig_b(b)$.

(6) We also say that $NameSpace_B$ refines $NameSpace_C$ if and only if:

1. $card(NameSpace_C) > card(NameSpace_B)$ *and,*
2. $\forall\, a \in NameSpace_B\ \exists b \in NameSpace_C, Sig_B(a) = Sig_b(b)$.

(7) C and B are said to be equivalent if and only if:

1. $NameSpace_C$ and $NameSpace_B$ are equivalent ; and,
2. $Keywords_B \cap Keywords_C \neq \emptyset$.

(8) We say that the context C refines context B if and only if:

1. $NameSpace_C$ refine $NameSpace_B$; and,
2. $Keywords_B \cap Keywords_C \neq \emptyset$.

3.2.2 Definition of the Description

The "description" part of any business rule is reserved for the specification guidelines that must be carried out to achieve the expected results. Formally, we define the description part of a business rule by the triplet:

$$Description : (\text{Guard}, Sequence, Results)$$

Where:

- *Guard*: is the condition to be satisfied in order for actions to be executed;
- *Sequence*: is a sequence of actions separated by commas;
- *Results*: is a specification of the expected result.

3.2.2.1 Guard

The guard is a condition defined in [8] which must be satisfied in order for the description part of the business rule to be activated. It is specified by a literal expression of human language featuring objects and attributes listed in the context. Concretely, it is materialized in the form ***"if condition then ... else ..."*** to *"condition"*. A guard shall be said to be simple if and only if it is the condition in the formulation ***"if condition then ...."*** It is said to be complex if and only if it is the condition in the formulation ***"if condition then ... else ...".***

A condition is an expression of natural language that expresses a reality that can be either true or false. To express expressions, we defined a number of predicates, of: comparison, coordination, and negation. The following table summarizes the predicates raised.

Table 1: list of comparison operators

Types of predicates	predicates	Notation
comparison predicates	…. Is less than….	(lt, ...,...)
	…. Is less than or equal to….	(le,...,...)
	…. Is greater than….	(ut,...,...)
	…. Is greater than or equal to…..	(ue,...,...)
	…. Is equl to…..	($\stackrel{def}{=}$,... ,...)
Conjunction predicates	…. and ….	($\wedge$,...,...)
	…. or ….	($\vee$,...,...)
Negation predicates	Opposite of….	($\neg$,...)

Conjunction predicates apply only to conditions. This is a conjunction of conditions using the conjunctions ('or' or 'and'). To reduce misunderstandings between the business executives and developers, it was important to bring our specification closer to natural language. It should be noted however that only objects declared in the context and without the mention "control" should be used. The predicates we define raised three types of expression: $CompExpression$ reserved for literal expressions depicting two objects of the context associated with a comparison predicate; $NegExpression$, reserved for expressions expressing the opposite of the reality expressed by a condition. $NegExpression$ applies only to conditions; $CrdExpression$, reserved for expressions which translate the composition of conditions. The formal representation of each of the preceding expressions is as follows:

$CompExpression = \langle Object_1.attr, Object_2.attr, \hat{} \rangle$

$NegExpression = \langle Cx_1, \neg \rangle$

$CrdExpression =< Cx_1, Cx_2 \mid Cr \mid NegExpression, \& >$

Where:

-$Object_x.attr$, represents an attribute of the $Object_x$, specified in the suffix *"views";*

- Cx_x is an expression of type $Expression$;

-Cr is an expression of type $xpression$;

-$\hat{}$ is a comparison predicate;

- $\&$ conjunction predicate;

-'$\neg$' negation predicate.

In general, a condition will be represented formally by:

$$Condition :< CompExpression \mid CrdExpression \mid NegExpression >$$

We denote by $Conditions$, the set of conditions and we define the function $TriCond: Conditions^2 \rightarrow Conditions$ such that for every pair (a, b) of $Conditions^2$, $TriCond(a, b)$ returns a sorted condition in the same sense as the above defined Tri function, if the terms a and b are bound by the same conjunction. $TriCond(a, b)$ returns a in other cases. Consider a condition β, we denote $Termes(\beta)$, the set of predicates comprising the condition β. We shall denote by $squelette$, the function $Squelette: Expressions \rightarrow String$, which for any expression a, $Squelette(a)$ returns a string obtained by replacing the various attributes of the objects composing this expression by their types without affecting the constants.

Axiom 5: Equivalence of Conditions

Consider two conditions a and b, we say that a and b are equivalent if and only if:

(10) $Squelette(a) = elette(b)$;

(11) one of the $Squelette$ in the reverse order of the other.

Axiom 6: Refinement or Extension of a Condition

(12) Consider two conditions a and b, we say that a is a refinement of b if and only if
$SubStr(squelette\ (a), squellete(b))$ is true.

(13) if (10) is satisfied, we say that b is an extension a..

In the following, the notion of $squellete$ will be extended to basic predicates included in the sequences.

3.2.2.2 Sequence

A *sequence* is a series of actions that must be executed to produce the result of the business activity. An action is a predicate, translating an atomic process or operation for achieving a partial or final objective. It modifies the state of the environment it is an expression with side effect. Two types of actions exist: simple actions and complex ones. Simple actions are those consisted of a single base predicate. Complex actions are those that refer to a set of base predicates, or business rules existing via the deterministic choice operator relative to a condition. The base predicates, inspired by [8], taken into account in our specification are:

- addition of... to ...;
- the withdrawal of .... from ....;
- the product .... by ... ..;
- Division ... .. by ... ..;
- editing of .... on ....;
- recording of .... in ....;
- the creation of ... .. in ....;
- the classification of ... .. in ....;
- the next of ... in ....;
- the previous of ... .. in ....;
- the update .... by ....

The set of base predicates constitutes the domain vocabulary. This concept was well presented in [8]. We will, therefore, not return to. However, it should be noted that extensions of the latter can be done based on natural language and the domain, to cover the set of atomic actions of this human language. Nevertheless, we recommend, basing on tests performed on the specification of business rules in the field of career management of State personnel and payroll, in a sample of twenty-five administrations, in Cameroon, that we maintain these predicates in state. However the modification of the number of arguments is permitted.

In human language, several twists of language can translate these predicates. It's for the software engineer assisted by the business executive to: specify the business rule, identify the comments of the business executive, the operation in question. To facilitate the expression of the business rule, we have, at the risk of repeating ourselves, chosed predicates which to some extent may reflect the same action. These predicates must put to stage some objects declared in the suffix "*aggregate*". Besides these basic predicates, we defined a predicate of deterministic choice denoted [...] {...}, as follows: *[obs$_1$,...,obs$_n$]{A$_1$,...,A$_m$}* where A_i is in the form *action$_i$(val.obs$_{i,1}$,...,val.obs$_{i,n}$),* and *val.obs$_{i,k}$* represents the value of *obs$_k$* for the action *i, and obs$_i$* denotes the property of an object declared in the suffix *aggregate.*

The business rule $action_i$ shall be executed if and only if the values of the parameters $Obs_1, Obs_2, \ldots, Obs_n$ corresponds to those of $val\_Obs_{i,1}, \ldots, val\_Obs_{i,n}$. This predicate enables execution under certain conditions, of a single action from among those listed. It also enables us to represent a complex action of multiple-choice. Formally, the "sequence" will be represented by:

$Sequence : (SimpleAction \mid ComplexAction)^+$

where :

-$SimpleAction$: is an action consisted of an unique basic predicate;

- $ComplexAction$:
$\langle SimpleAction \mid [\ldots]\{\ldots\} \mid [, ComplexAction] \rangle$.

We denote by $Sequences$, the set of business process sequences.

Axiom 7: Equivalence of actions

Two actions a and b are said to be equivalent if and only if :

$$squelette(a) = squelette(b).$$

We define the function $processStr$ as follows: $processStr : Sequences \times NameSpace_C \rightarrow SimpleType$ such that $\forall (s, o) \in Sequences \times NameSpace_C C$, $processStr(s, o)$ is the set of processes which object o is subjected to in the sequence s. $processStr(s, o)$ is also called the modifications string of the object o. $processStr(s, o)$ materializes the effect of processes on the object o.

Axiom 8: Equivalence between Sequences

Consider two sequences s_1 and s_2, we say that s_1 and s_2, are equivalent if and only if : $\forall a \in NameSpace_C$, $processStr(s_1, a) = processStr(s_2, a)$

Axiom 9: Extension, Refinement of Sequence

Consider two sequences s_1 and s_2, we say that s_1 is a refinement of s_2 (or s_2 is an extension of s_1) if and only if the s_2 contains the predicate of deterministic choice and there exist a sequence s in this predicate, such that, if a is an element of $NameSpace_C$, then

$$processStr(s_1, a) = processStr(s, a)$$

3.2.2.3 Results

The result is what we observe at the end of the business activity. Results indicators are defined in the same way as for attributes of object that are used in the description part of a business rule. In a formal way, let *Ob* be an object, *F* a given field of *Ob* of a given simple type *ST*, a result Rt is defined as follows:

$$Rt = (Ob.views\{\ F\ \%ST[, F\%ST]^*\})^+$$

Meanwhile, in the result specification of results, certain attributes of objects may not contain the mention *"reference"*. We denote by $ResultObjets$, the set of results objects of a business rule and $card(ResultObject)$ the number of objects in this set. Elements of $ResultObjets$ implicitly have the mention *"artifacts."* The results can also be seen as the goal to attain.

Axiom 10: Equivalence Results

Consider two results a and b, we denote $ResultObjets_a$ ($ResultObjets_b$ respectively), the set of result objects of a (of b respectively) we shall say that a and b are equivalent if and only if:

1- $card(ResultObjets_a) = card(ResultObjets_b)$ and,

2- $\forall o_1 \in ResultObjets_a$,
$\exists o_2 \in ResultObjets_b, Sig_a(o_1) = Sig_b(o_2)$.

Axiom 11: Extension, Results Refinement

Consider two results a and b, we denote $ResultObjets_a$ (respectively $ResultObjets_b$) the set of result objects of a (of b respectively), we say that a is a refinement of b, if and only if:

1- $card(ResultObjets_a) > card(ResultObjets_b)$ and,

2- $\forall o_1 \in ResultObjets_a$,
$\exists o_2 \in ResultObjets_b, Sig_a(o_1) = Sig_b(o_2)$.

After presenting the different parts of a business rule, we formally define the business rule as follows:

$$rule\_name = (Contexte_{rule}, description_{rule})$$

where: - $Contexte_{rule}$: represents the context part of the business rule

- $description_{rule}$: represents the description part o the business rule

The formal representation of the context and description parts of a business rule were given in the previous sections. We also defined a number of relationships such as equivalence, refinement between concepts developed in the sections above. As we proceed, we shall use these relations to develop relationships between the business rules.

4. RELATIONSHIP BETWEEN BUSINESS RULES

In the previous section, we have defined a set of concepts and relationships between these different concepts. All these works were intended to clearly present our vision of the business rule and prepare the ground for defining relationships between different business rules. The lines that follow shall be devoted to these relationships. Furthermore, we denote equivalence between two concepts by: ≡; refinement by: ≅, and $views(b, \alpha)$, all attributes of the object b in the business rule α.

4.1 Axioms

Consider two business rules α and β, we denote by $Contexte_\alpha$, $Contexte_\beta$ the respective contexts of the business rules α and β and $Description_\alpha = (Garde_\alpha, Sequence_\alpha, Result_\alpha)$, $Description_\beta = (Garde_\beta, Sequence_\beta, Result_\beta)$ the descriptions part of the business rule α and β respectively.

Axiom 12: Equivalence of Business Rules

We say that α and β are equivalent if and only if:

$$Contexte_\alpha \equiv Contexte_\beta \ et \begin{cases} \text{Sequence}_\alpha \equiv \text{Sequence}_\beta \\ \text{Garde}_\alpha \equiv \text{Garde}_\beta \\ \text{Result}_\alpha \equiv \text{Result}_\beta \end{cases}$$

Axiom 13: Extension, Refinement of Business Rules

We say that α is a refinement β or that β is an extension of α if and only if:

$$Contexte_\alpha \cong Contexte_\beta \ and \begin{cases} \text{Sequence}_\alpha \cong \text{Sequence}_\beta \\ \text{Garde}_\alpha \cong \text{Garde}_\beta \\ \text{Result}_\alpha \cong \text{Result}_\beta \end{cases}$$

Axiom 14: Inconsistent Business Rules

We say that a business rule is inconsistent, if and only if:

1- there exists at least one object in the context of this business rule that is not used in the description of that business rule;

2- there exists at least one attribute or property of an object from its context which is not used in the description part of that business rule;

Axiom 15: Incomplete Business Rules

A business rule is said incomplete if and only if it is not inconsistent and there are properties that have the mention "reference" which does not appear in the result objects of the same rule.

Axiom 16: Merging Business Rules

We say that two business rules α and β can merge if and only if:

1- α and β are neither inconsistent nor incomplete;

2- $Keywords_\alpha \cap Keywords_\beta \neq \emptyset$ and $NameSpace_\alpha \cap NameSpace_\beta \neq \emptyset$

3- $\exists\ b \in NameSpace_\alpha \cap NameSpace_\beta$ and $views(b, \alpha) \cap views(b, \beta) \neq \emptyset$;

4.2 Impact of these Relations on the Business Process Requirements Model

Definition (2)***:*** Sequencing Rule

A business rule α is a sequencing rule if $Sequence_\alpha$ contains the predicate of deterministic choice.

In [1], we presented a goal oriented approach- for the definition of a business process requirement model, taking into account their level of importance and constraints inherent to these requirements. The level of importance of a goal is the credit which the user associates to this goal. Constraints are non-functional requirements related to what this goal must satisfy. The approach that was proposed in [1], revolves around four main activities: requirement elicitation, selection of different goals, transformation of requirements into knowledge bits and finally the development of the requirement model. We have shown formally that this approach will exhaustively describe a business process. To do this, we have given a formalism to

model the requirements of a business executive and deduced from the work of [4], a formal representation of what we call knowledge bit or expressed requirement. An expressed requirement or knowledge bit was defined as follows:

$$\partial = (\psi, \omega, \lambda, \delta, \nu)$$

where :
∂ *name of* knowledge bit
ψ *is the context in which the gola is defined*
ω *is the goal*
λ *is the business rule*
δ *represents contraints*
ν *is the level of importance of the goal*
∂ is the name of a domain concept ψ.

Let's consider $a = (\psi, \omega, \lambda, \delta, \nu)$ and $b = (\psi', \omega', \lambda', \delta', \nu')$ two expressed requirements S^a is the set of objects of the organizations' information system, for which the expectation $a.\omega$ is satisfied under the rule $a.\lambda$ and the constraint $a.\delta$ [1]. It is the same for S^b

The concepts of requirements identity, sub-requirements, and sub division of requirements were clearly defined in [1]. This definition was exclusively focused on usage intension. We shall not return to this. We shall use these characteristics to show the impact of a business rule on the organizations' business processes requirement model.

Impact 1: $S^a = NameSpace_{a.\lambda}$ by definition.
Impact 2: $a.\lambda \equiv b.\lambda'$ therefore a *and* b are identical

Proof: We assume $a.\lambda \equiv b.\lambda'$ and shall show that $\psi = \psi'$, $\omega \approx \omega'$ et $S^a = S^b$

Consider two requirements a and b, in the conditions of the paragraph above, we assume that $a.\lambda \equiv b.\lambda'$, by definition of $a.\lambda \equiv b.\lambda$ we have:

a) $Contexte_{a.\lambda} \equiv Contexte_{b.\lambda'}$, that is $NameSpace_{a.\lambda} \equiv NameSpace_{b.\lambda'}$ and $Keywords_{a.\lambda} \cap Keywords_{b.\lambda'} \neq \emptyset$, hence, $S^a = S^b$;
b) $Results_{a.\lambda} \equiv Results_{b.\lambda'}$, by definition, $\omega \approx \omega'$.
c) From a) and b) we deduce that $\psi = \psi'$.

Impact 3 : $a.\lambda \cong b.\lambda'$, a is a sub-requirement of b.

Proof: Suppose that $a.\lambda \cong b.\lambda'$. 'and show that $S^a \subset S^b$ and $b = \rho^a$. Consider two requirements a and ; by $\lambda \cong \lambda'$ in the conditions in the above paragraph, we assume that $a.\lambda \cong b.\lambda'$; by definition of $a.\lambda \cong b.\lambda'$ we have: $\lambda \cong \lambda'$ definition of a:

a) $Contexte_{a.\lambda} \cong Contexte_{b.\lambda'}$, that is to say $NameSpace_{a.\lambda} \cong NameSpace_{b.\lambda'}$ and $Keywords_{a.\lambda} \cap Keywords_{b.\lambda'} \neq \emptyset$. $NameSpace_{a.\lambda} \cong NameSpace_{b.\lambda'}$ we deduce by definition that $S^a \subset S^b$;
b) $Séquence_{a.\lambda} \cong Séquence_{b.\lambda'}$ by definition the $a.\lambda$ is referenced in $b.\lambda'$ therefore $b = \rho^a$.

Impact 4: (definition of divisible requirement):
We say that a is divisible if and only if $Sequence_{a.\lambda}$ contains the predicate of deterministic choice. The number of business rules referenced in the predicate of deterministic choice constitute the number of parts of requirement a.

Impact 5 (definition of ambiguous requirement):
We say that an expressed requirement a is ambiguous if and only if $a.\psi$ is not in the list of domains listed in the domain of operations of the business rule.

Impact 6 (definition merging requirements*):*
We shall say that two requirements a and b can merge if and only if $a.\lambda$ is merged to $b.\lambda$.

These impacts allow us to complete the definition of the concepts discussed in [1] which remained superficial.

4.3 Business Object Model (BOM)

We mean by "Business Objects", an object referenced in a business rule. Business objects are manipulated in the description part of a business rule. This part enables one to describe exhaustively the various business objects of business processes. In this section, we present the methodology of defining business objects.

Consider $R_{BP} = \bigcup_i^n b.\lambda$ (where b is an expressed requirement, λ the business rule of b), the set business rules of a business process $NameSpace_{BP} = \bigcup_{i=1}^n NameSpace_{r_i}$ (where r_i is the i^{th} business rules; $NameSpace_{r_i}$ is the set of objects referenced in the rule r_i), of objects referenced in all business rules; A_{BP}, all the business objects attributes, a function $prop: NameSpace_{BP} \times R_{BP} \rightarrow A_{BP}$ that for every pair (a,r) of $NameSpace_{BP} \times R_{BP} \rightarrow A_{BP}$, $prop(a,r)$ returns the set of attributes of the object a referenced in the rule r .We define a function $fields$: $NameSpace_{BP} \rightarrow A_{BP}$, such that if $b \in NameSpace_{BP}$, then

$$fields(b) = \bigcup_{r \in R_{BP}} prop(b,r).$$

Property 1: Business Object Attributes
Consider a business object O, of $NameSpace_{BP}$, $fields$ represents the set of attributes of the object O.

Let *field*^{ref} denotes a function *field*^{ref}*:* $NameSpace_{BP} \rightarrow A_{BP} \times NameSpace_{BP}$, the attributes of an object $b \in NameSpace_{BP}$ referenced in a business rule is given by :

$$fields^{ref}(b) = \bigcup_{r \in R_{BP}} prop^{ref}(b,r).attributs_b$$

Where:
- $prop^{ref}: NameSpace_{BP} \times R_{BP} \rightarrow A_{BP} \times NameSpace_{BP}$ is a function for every pair (b,r) of $NameSpace_{BP} \times R_{BP}$, $prop^{ref}(b,r)$ returns the set of pairs $(attribut_b, a)$ such that $attribut_b$ represents the set of attributes of the object b having the mention "*reference"* in the suffix "*views"* of the object of a business rule r.

Property 2: *Reference Attributes of a Business Objects*
Consider a business object O of $NameSpace_{BP}$, $fields^{ref}(O)$ represents the set of attributes of the object O with the mention "*reference"* in the suffix "views" of business objects of $NameSpace_{BP}$.

Let $fields^{tag}$, be a function $NameSpace_{BP} \rightarrow A_{BP} \times NameSpace_{BP}$, the attributes of an object b referenced in the business rule is defined by :

$$fields^{tag}(b) = \bigcup_{r \in R_{BP}} prop^{ref}(b,r).a$$

Where:

- $prop^{ref}: NameSpace_{BP} \times R_{BP} \rightarrow A_{BP} \times NameSpace_{BP}$ is a function for every pair (b,r) of $NameSpace_{BP} \times R_{BP}$, $prop^{ref}(b,r)$returns the set of pairs $(attribut_b, a)$, such that $attribut_b$ represents the set of attributes of the object b having the mention "reference" in the suffix "views" of the the business rule object r.

Property 3: Links between business Objects
Consider a business object O of $NameSpace_{BP}$, $fields^{tag}(O)$ represents the set of business objects with references in the attributes of O. These references are called links or connections between O and other business objects.

***Constraints**: (uniqueness of the name business object)*

The name of a business object is unique in a business processes. It can be reused as many times as you want in the context part of different rules. It is the same for attributes of these business objects. The latter retains their types and their names, whatever the business rule.

5. CONCLUSION AND FUTURE WORKS

After this work which focused on formal specification of business rules of a business process, we introduced a formalism to formally describe business rules of a business process. The approach that we proposed in this paper firstly presents a formalism for the specification of business rules, and then extends the notion of: equivalence, refinement, and extension to these business rules through a range of axioms. These axioms are based on a number of predicates which themselves are black boxes. A the end, we present the impact of this specification on the concepts of requirements identity, sub-requirements, sub division of requirements and the compositions of requirements.

However, we were not interested on operations contained in base predicates. We thought of this as subject of research in our laboratory. We envisage in the coming days:

- to Define passage rules from business process requirement model to a business component model;
- the introduction of performance indicators in the modeling of business processes;
- to define a platform for identifying a system requirements model and in the same urge identify reusable requirements;
- to enrich the work on selection of software components.

The purpose of all this work is to implement a component-based development platform from requirement specification closer to human language. This shall surely permit us to minimize misunderstandings between developers and business executives, and to produce systems on the basis of software components of lower costs while mastering the changing requirements in a business process.

6. REFERENCES

[1] R. Atsa Etoundi, M. Fouda Ndjodo, Christian Lopez Atouba, "A Goal Oriented Approach for the Definition of a Business Process Requirement Model", IJCA, 2010.

[2] R. Atsa Etoundi, M. Fouda Ndjodo, « Human Resource Constraints driven Virtual Workflow Specification », IEEE SITIS pp 176-182, 2005.

[3] R. Atsa Etoundi, M. Fouda Ndjodo, « Feature-Oriented Business Process and Workflow », IEEE SITIS pp 114-121, 2005.

[4] Farida Semmak, Joël Brunet, « Un métamodèle orienté buts pour spécifier les besoins d'un domaine », 23e Congrès INFORSID, pp 115-132, mai 2005.

[5] Lubars, M., Potts, C., Richer, C.: A review of the state of the practice in requirements modeling.Proc. IEEE Symp. Requirements Engineering, San Diego 1993.

[6] Karen Mc Graw, Karan Harbison, User Centered Requirements, The Scenario-Based Engineering Process. Lawrence Erlbaum Associates Publishers, 1997.

[7] The Standish Group, Chaos. Standish Group Internal Report, http://www.standishgroup.com/chaos.html, 1995

[8] Mouhamed Diouf, « Spécification Et Mise En Œuvre D'un Formalisme De Règles Métier» , thèse n°3507, Université Bordeaux I, décembre 2007.

[9] "Anonyme" Business Semantics of Business Rules, *Business Rules Team Response to RFP: SBVR Submission bei*/2005-08-01 Version 8. http://www.omg.org/docs/bei/05-08-01.pdf last accessed on 2007/05/02

[10] "Anonyme" Semantics of Business Vocabulary and Business Rules Specification, Document de specification de l'OMG, 2006. http://www.omg.org/docs/dtc/06-03-02.pdf last accessed on 2007/05/02

[11] Vincent Legendre, Gérald Petitjean , Thierry Lepatre, « Gestion des règles métier », Génie Logiciel n° 92 mars 2010, pp 43-52, 2010

[12] Tim van Eijndhoven, Maria-Eugenia Iacob, María Laura Ponisio "Achieving Business Process Flexibility With Business Rules", 12th IEEE EDOCC, pp 95-104,2008.

[13] Object Management Group, *Production Rule Representation: Request for Proposal*, br/2003-09-03, Sept. 2003. http://www.omg.org/docs/br/03-09-03.pdf.

[14] David Hay and Keri Anderson Healy. De_ning Business rules What Are They Really ? Technical Report 1.3, The Business Rules Group, July 2000

[15] Ronald G. Ross. Principles of the Business Rule Approach. Addison-Wesley, Boston, USA, 2003.

[16] Business Rule Group Community BRG. http ://www.brcommunity.com. Web page.

[17]D. Hay and K. A Healy. "GUIDE Business Rule Project Final Report". Technical report, 1997.

[18] The Object Management Group OMG, UML 2.0 OCL Specification.OMG Specification, October 2000