A rule-based approach to model and verify flexible business processes

Mohamed Boukhebouze*

PReCISE Research Center,
University of Namur,
rue de Bruxelles 61, B-5000 Namur, Belgium
E-mail: mohamed.boukhebouze@fundp.ac.be
*Corresponding author

Youssef Amghar and Aïcha-Nabila Benharkat

CNRS, INSA-Lyon, LIRIS,
Université de Lyon,
UMR5205, F-69621, France
E-mail: youssef.amghar@insa-lyon.fr
E-mail: nabila.benharkat@insa-lyon.fr

Zakaria Maamar

College of Information Technology,
Zayed University,
P.O. Box 19282, Dubai, UAE
E-mail: zakaria.maamar@zu.ac.ae

Abstract: Efficient organisations need to ensure that their business processes are flexible so that these processes can easily accommodate changes in regulations and policies. Appropriate techniques to model and verify these processes are required. In this paper, we present a rule-based approach, which is built upon the event-condition-action model (ECA) and supported by a rule-based business process definition language (RbBPDL). In this approach, rules, which specify business processes, are represented using the event-condition-action-post-condition-event (ECAPE) model. This allows translating a process into a graph of rules that is used to check how flexible a business process is, and estimating this process's cost of changes. This cost is based on a rule change cost model (R2CM) that will be presented in this paper. In addition, the ECAPE model allows the translation of a process into a coloured Petri net, called ECAPE net, in order to verify process functioning prior to any deployment.

Keywords: business processes modelling; reaction rule; declarative language; rule graph; flexible modelling; business processes verification.

Reference to this paper should be made as follows: Boukhebouze, M, Amghar, Y., Benharkat, A-N. and Maamar, Z. (2011) 'A rule-based approach to model and verify flexible business processes', *Int. J. Business Process Integration and Management*, Vol. 5, No. 4, pp.287–307.

Biographical notes: Mohamed Boukhebouze is currently a Post-Doctoral Researcher at the Faculty of Computer Science of the University of Namur. His current research interests include business process management and web services.

Youssef Amghar is a Professor at the University of Lyon. His current research interests include business process, and interoperability of applications.

Aïcha-Nabila Benharkat is an Associate Professor at the University of Lyon. Her research interests include the schema matching techniques, as well as the interoperability in information system.

Zakaria Maamar is a Professor at the College of Information Technology, UAE. His research interests include wireless information systems, ubiquitous computing and web services.

1 Introduction

1.1 Motivations

To remain efficient and competitive, organisations need to make sure that their business processes are flexible. The development of these processes requires appropriate models and tools that would permit defining, deploying, and controlling these processes. Business process management (BPM) is about managing the whole cycle of a business process from early stage of requirement elicitation to later stage of deployment and maintenance. In this life cycle, process modelling sheds the light on the best practices and core knowledge in an organisation. For this purpose, this modelling must be based on sound languages that provide the right means to describe the various elements of a process, to support the flexibility of a process by changing only the parts that need to be changed while keeping other parts stable (Regev et al., 2006), and to support verification to ensure process reliability. However, these characteristics are highly inter-related, which increases the complexity of satisfying them all at once. Real business processes tend to be less flexible and difficult to analyse due to continuous changes in regulations and policies.

1.2 Issues

In the literature, imperative business process modelling languages such as business process modelling notation (BPMN – OMG, 2009) focus on how the various activities in a process are ordered during execution. These languages provide a good level of expressivity and several verification techniques (e.g., Petri nets, PNs) have been proposed to ensure the reliability of the processes that are defined using imperative languages. However, the use of these languages forces designers to describe explicitly the execution scenarios (pre-computation of task control flows, information flows, and work allocation schemes) in the modelling phase, which is not very convenient. This makes business processes rigid and difficult to maintain, which is not in line with the dynamic nature of organisations.

Rule-based languages have been proposed to deal with the flexibility requirement in a proper way by modelling the logic of a process with a set of rules that comply with declarative languages' guidelines. Examples of these languages include PENELOPE (Goedertier and Vanthienen, 2006), DECLARE (Pesic et al., 2007) and standard semantics of business vocabulary and business rules (SBVR) (OMG, 2008). Therefore, the execution scenarios are implicitly done in the modelling phase. This could avoid listing all the possible scenarios of execution in the modelling phase, which is difficult to obtain (Goedertier and Vanthienen, 2006) and can result in rigid models. However, the verification stage is more complex since it requires having an execution scenario that ensures the proper functioning of a process. In addition, automating the process of developing execution scenarios using an implicit declarative model is not clear (Goedertier and Vanthienen, 2006).

1.3 Contributions

In this paper, we present a rule-based approach that aims at improving the management of business processes in terms of flexibility and verification. This model extends the event-condition-action (ECA) model and suggests formal tools for verification purposes. In this approach, the logic of a process is defined with a set of business rules that correspond to the policies in the organisation. Each business rule is represented using the event-condition-action-post-condition-post-event (ECAPE) model (or formalisms). An advantage of the ECAPE formalism is that a process can be easily translated into a graph of rules. This graph is used to first, look into the flexibility of a process by checking the relationships between the rules and second, estimate cost changes in a process using our rule change cost model (R2CM). Another advantage of the ECAPE formalism is the translation of a process into a new coloured PN called ECAPE net. An ECAPE net is used to check if a process satisfies some properties such as no Deadlock, and no Livelock. The representation of our rule-based approach requires a new declarative language that will offer the necessary syntax and semantics to describe ECAPE rules and the core elements in a business process. These elements are participants, variables, and activities. For this reason, we propose the rule-based business process definition language (RbBPDL), which has an XML-based syntax to describe business processes in declarative way.

1.4 Paper organisation

The rest of this paper is organised as follows. Section 2 contains definitions on some basic concepts and states the particular issue that is addressed. Sections 3 and 4 detail our proposed approach to model business processes. Section 5 discusses process change management and impact change estimate. Section 6 explains how a process is verified in our approach. Section 7 describes an architecture that combines the different techniques and tools presented in this paper. The last sections wrap up the paper in terms of related work, conclusions, and some directions for future works.

2 Preliminaries

Prior to stating the specific issues that are dealt with in this paper, a set of definitions on business rule, process flexibility, and process verification are given.

2.1 Definitions

2.1.1 Type of rules

To carry out their missions, organisations manage complex processes and need to react to changes. According to Goedertier and Vanthienen (2006), changes could be due to new or revised regulations and policies that organisations have to comply with. These regulations and policies are often expressed in terms of rules.

Rules are defined as high-level structured statements that constrain, control, and influence the business logic of a process (BRG, 2002). According to Wagner (2005), a business rule could be categorised as follows:

1 Integrity rule identifies constraints or assertions that must be satisfied (e.g., a customer must be registered before submitting any order).

2 Derivation rule concerns one or more conditions and one or several conclusions (e.g., a loyal customer receives a 10% discount; Boukhebouze is loyal; therefore, Boukhebouze receives a 10% discount).

3 Production rule concerns one or more conditions and one or more actions (e.g., if the stock is low, then execute the supply process).

4 Reaction rule consists of events to occur, conditions to satisfy, and actions to execute (e.g., upon order receipt and if the necessary raw materials are available, then start production).

5 Transformation rule controls changes in a system state (e.g., an employee's age must be changed in an incremental way).

It should be noted that Deontic assignment rule can be considered as another category of business rule (Taveter and Wagner, 2001). This rule constrains actions in terms of obligations, permissions, and prohibitions (e.g., only the manager has the right to grant promotions to employees).

Rules should be formalised to facilitate their use. Unfortunately, using imperative languages such as BPMN forces designers to implement rules based on decisions (what process branch must be chosen) that are defined with connectors (e.g., sequence, parallel split, exclusive choice). Through these languages, decisions' results determine a process behaviour rather than how these decisions should be modelled. Consequently, implementing changes in a process turns out complex and prone to errors.

2.1.2 Process flexibility

By flexibility, we mean how changes are implemented in some parts of a business process without affecting the rest of parts neither the continuity nor the stability of these parts (Regev et al., 2006). In a dynamic environment, businesses need to rely on flexible process models to allow a controlled modification of some parts of a process. According to the taxonomy of Regev et al. (2006), all the elements of a process are likely to be changed. A change may affect a process's activities (functional aspect), a control flow (behaviour aspect), a process data (information aspect), or protocols used in a process (operational aspect). However, a change in a process element may require changes in other elements and so on for the sake of guaranteeing process coherence. To do so, the impact of a change on the rest of the process should be examined and estimated.

2.1.3 Process verification

By verification, we mean how to ensure that process execution happens in accordance with its modelling plan and how to detect unanticipated situations that may arise during the execution of this process (Russell et al., 2006). An erroneous business process has major negative consequences on its continuity. To assist a designer detect errors during process specification, several techniques are used. The verification by formal models is widely used (e.g., PN, process algebra, etc.). There exist a good number of works that use formal models to verify the proper functioning of business processes (Ouyang et al., 2005; Yang et al., 2005; Koshkina and Van Breugel, 2004; Pu et al., 2005).

2.2 Problem statement

In this paper, we propose a rule-based approach to model the logic of a process with a set of rules that comply with declarative languages' guidelines. This way of doing allows deploying partially-specified process definitions (Lu and Sadiq, 2007). A rule engine determines, at runtime, what to execute by evaluating relevant rules with regard to a certain process event. According to Lu and Sadiq (2007), a rule-based approach externalises the process logic from the execution environment. Consequently, the modifications in a process definition can be made without impacting the executing process instances. In addition, the changes (in process logic, business regulations, or business policies) are implemented by changing a subset of rules (e.g., modify, insert and delete existing rules), which express the changed process logic, the changed business regulations, or the changed business policies. As a result, the modification in a rule impacts only a subset of rules that are related to the changed rule, which would lead to a decrease of the efforts to put into this change management.

However, when it comes to complex processes, it is important to manage the impact of a rule change on the rest of the processes by determining which rules are impacted by this change and estimating the overall cost of this change. Although the evaluation of business process changes impact is not trivial and should be carefully examined, this evaluation is beneficial when several change alternatives are offered and planning, organising, and managing resources to ensure the success of these changes.

The research questions that are raised here concern: what is the rule formalism that would offer better support to change management impact, and how is this change impact estimated so that BPM in terms of modelling and verification is improved?

3 Rule-based modelling of business processes using the ECAPE model

The objective of a rule-based approach is to describe business processes with focus on their behaviours using a set of connected rules. According to Giurca et al. (2006), it

is advantageous to use reactive rules (ECA formalism) for business process specification. Giurca et al. argue that rules along with their events give a flexible way to specify a process's control flow. In addition, ECA rules are easier to maintain and include other types of rules such as integrity and derivation. However, the ECA formalism does not cover the execution control and does not allow having an explicit execution scenario that is needed to verify the proper functioning of a process. To this end, we propose to extend the ECA formalism initially defined by event, condition, action with post-condition and post-event. The ECAPE formalism is defined as follows:

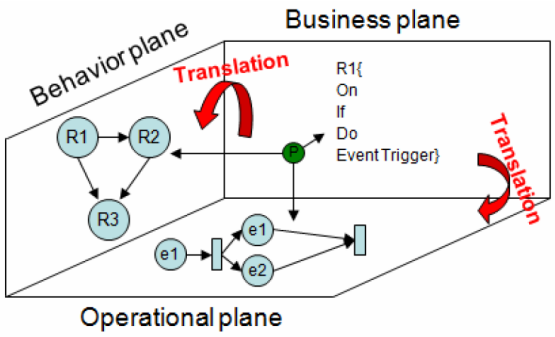
ON	<Event>
IF	<Condition>
DO	<Action>
Check	<Post condition>
Raise	<Post event>

The semantics attached to an ECAPE rule is: event determines when a rule must be evaluated (or activated); condition is a predicate upon which the execution of an action depends (it can be seen as a refinement of the event); action specifies the code to execute if the condition is satisfied; post-condition is a predicate upon which the validation of the rule depends (the rule is validated only if the post-condition is satisfied); and event-triggered (post-event) identifies the set of events that arise after the execution of the action. Note that, if a post-condition does not hold, cancellation mechanisms are executed in order to cancel, if possible, the effects of the executed action. These mechanisms do not fall within the scope of this paper.

A sequence of ECAPE rules defines the behaviour of a process. Each rule may activate one or more rules. The originality of the ECAPE formalism is that the set of events triggered after the execution of a rule's action, is explicitly described. As a result, a sequence of rules can be automatically deducted.

In Figure 1, we show how we look at a business process from three different abstract plans:

Figure 1 Three plans of our rule-based modelling approach (see online version for colours)



1 *From a business plane*, business processes are defined with a set of ECAPE rules. These rules express, in a declarative way, a business domain, policies, business concerns, just to cite a few. We propose the RbBPDL to formalise these rules.

2 *From a behaviour plane*, business processes are translated into a graph of rules where the vertexes represent the rules and the edges represent the relationships between these rules. This graph is used an input of assessing the cost of changes as defined in the R2CM.

3 *From an operational plane*, business processes are translated into an ECAPE net so that their proper functioning is verified.

In the rest of this paper, we describe the RbBPDL that is used to express business concerns, the graph of rules that are used to manage process changes and the ECAPE net that is used to express and verify the operational process.

4 The RbBPDL

To represent the ECAPE formalism as well as the various elements of a business process, we use RbBPDL that is XML-based and inspired by BPEL and XPDL [Figure 2(b)]. The overall structure of RbBPDL [Figure 2(a)] represents five elements of a business process (see Appendix 1):

1 *Participant* is any person, application, web service, or entity, which has the authority to perform an activity.

2 *Variable* represents pieces of information, which are produced or handled by a business process.

3 *Business activity* is the process element, which must be executed. It is a unit of work that runs in an indivisible manner by a participant.

4 *Event* is an indicator that signals that a situation has occurred and for which a response is necessary. The event is the activator element of a rule. It specifies when the rule should be evaluated. There are two categories of events:

   a Simple events describe the occurrence of a predefined situation in the system such as: activity events (start, end, cancellation, and error), process events (error trigger), time events (timer), and external events (reception of message signal).

   b Complex events that combine simple and/or composed events using constructors such as:

   - *Disjunction (e1, e2)* specifies that at least one of the two events is detected.
   - *Conjunction (e1, e2)* specifies that the events take place without taking into account their occurrence order.
   - *Occurrence (e1, nbr_occurence)* specifies multiple occurrences of the same event.
   - *Sequence (e1, e2)* specifies the sequence of events.
   - *Not (e1, t)* characterises.

5 *Business rule* is a statement that controls the logical relationships between the activities of a process, influencing the interactions between partners in this process. As rules are described using the ECAPE, they are represented as follows:

   a *OnEvent*: all events that activate a rule.

   b *Precondition*: predicates upon which the execution of an action depends.

   c *Action*: set of instructions to be executed if a precondition is satisfied. For this, we use a set of predefined instructions such as:

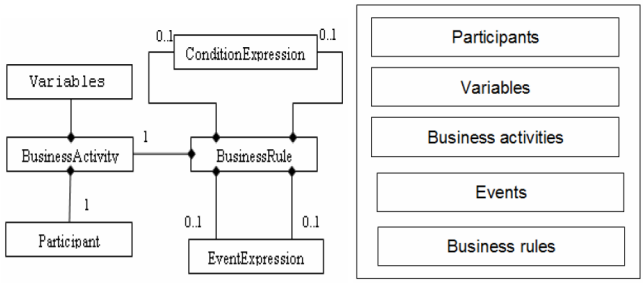
   - *Execute*: to execute the activity by a participant
   - *Cancel*: to cancel the execution of the activity
   - *Skip*: to skip the execution of the activity
   - *Discover*: to find a service that performs a given activity in a given registry

   d *Postcondition*: predicate upon which the validation of a rule depends.

   e *EventTriggered*: the set of events triggered by the execution of all the instructions in a rule's action.

To illustrate our rule-based modelling approach, we provide an illustrative example in the running example.

Figure 2 The global structure of the RbBPDL



(a) (b)

4.1 Illustrative example

In this section, we introduce the example of purchase order process to illustrate the ECAPE and the RbBPDL. Upon order receipt from a costumer, the calculation of the initial price of the order and selection of a shipper are done simultaneously. When both tasks are complete, a purchase order is sent to the costumer. If he accepts the purchase order, he will receive a bill for the needs of payment and registration. During this part of the process, two constraints need to be taken into account: customers must exist in the company database, and bills must be issued 15 days before delivery date.

Figure 3 represents the ECAPE rules set of the purchase order process. For example, rule R_1 expresses receiving an order. During the occurrence of 'receipt order' event, this rule is triggered and 'check if the customer is registered in database' action will be executed if 'the order is valid' condition is satisfied. The execution of this action is validated if 'database connection is correct' post-condition is true. Having a post-condition permits to control the execution of a rule's action. After the validation of an action's outcomes, 'costumer is checked' event is triggered. This latter activates three rules namely R_2 (initial price calculation), R_3 (shipper selection), and R_4 (reject order when costumer is not registered). Next, the execution of these rules' actions activates other rules, and so on until all the valid rules are executed. Through the explicit description of the events that are triggered after the execution of a rule's action, it is possible to deduct the sequence of rules, which permits to analyse a process. This is shown with the right arrow in Figure 3. Note that each rule has a post-condition, which permits to control the execution progress of a business process.

To describe the ECAPE rules of a business process, RbBPDL is used. Figure 4 represents an RbBPDL rule of the purchase order process. Participants [Figure 4(a)], variables [Figure 4(b)], business activities [Figure 4(c)], and events [Figure 4(d)] are represented with XML tags, i.e., 'participants', 'variables', etc. Additional details could be added to the representation of a rule so that a complete definition of a process's elements is offered. These details could concern the type and role of each participant, data types of each variable, input/output parameters of each activity, and type and constructors used to express the events.

In RbBPDL, the set of rules is described in <Business rules> part. For example, rule R1 expresses receiving an order [Figure 4(e)]. If (<OnEvent> $ Receive_Order </OnEvent>) event occurs, the rule is triggered to ensure that the information is valid (<PreCondition> $ info_Costumer! = "" </Precondition>). <Execute> action is executed and specifies that a given business activity must be performed (<Operation> $Costumer_Verification </Operation> in our example) by specifying the input/output parameters (<InputVariableName> $ info_Costumer </InputVariableName> and <OutputVariableName> $CostumerRegistration </OutputVariableName>). And also indicating which participant has the role to perform this activity (<Performer> $Commercial_Service </Performer>) of rule R_2 will trigger the events expressed in the <PostEvents> (in our example the event of costumer check termination $End_Costumer_Verification). Finally, rule R_2 is validated if the predicates of the post-condition expressed in the tag <PostCondition> are true (<PostCondition > $Database_connexion = true </PostCondition>).

However, taking into account the dynamic of the various process elements led us look for a better model to model business processes. For example, if the enterprise decides not to deliver its products, rule R_3 will be deleted from the process model with minimal impact on the rest of rules. Consequently, we deem appropriate to determinate the set of rules that would be impacted when rule R_3 is changed in order to keep the coherence of the process and estimate the overall cost of this change.

Figure 3 CAPE rules set of the purchase order

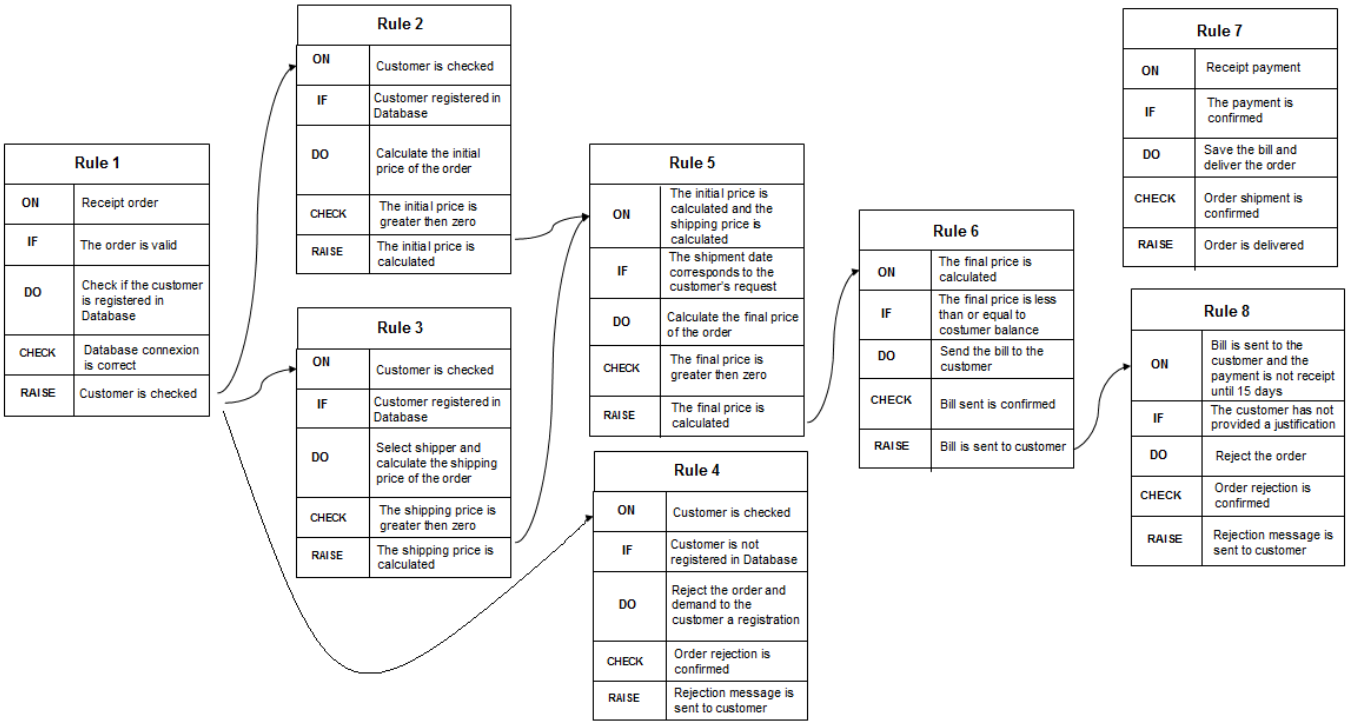


Figure 4 One rule of the RbBPDL purchase order process (see online version for colours)

```
<Participants>
  <ParticipantType>
    <SpecificResource name = 'FinancialApplication'>
      <ApplicationType>
        <DefinitionLanguage>WSDL</DefinitionLanguage>
        <CommunicationProtocol>SOAP</CommunicationProtocol>
        <Description>Serice Web  ...</Description>
      </ApplicationType>
      <ExternalReference>192.168.0.1</ExternalReference>
    </SpecificResource>
  </ParticipantType>
  <Role> Service financier  </Role>
</Participant>
```

(a)

```
<Variables>
<variable name="Info_Costumer">
  <DataType>
    <ArrayType LowerIndex ="0" UpperIndex ="50" >
      <DataTypes>
        <BasicType> STRING</BasicType>
      </DataType>
    </ArrayType>
  </DataType>
</variable>
```

(b)

```
<Events>
  <AtomicEvents>
    <AtomicEvent name="RecieveOrder"   EventGroup="StartGroup">
      <EventType>
        <MessageEvent>
          <Target/>
          <Source/>
        </MessageEvent>
      </EventType>
    </AtomicEvent>
  </AtomicEvent>
<CompositEvent name="ReplyToCostumer"
               EventGroup="EndGroup"
               ExpressionLanguage= "XPATH 1.0">
  <CompositExpression>
  Conjonction ($EndCalculateShippingPrice, $EndCalculateInitialPrice)
  </CompositExpression>
</AtomicEvent>
```

(c)

```
<BusinessActivities>
<BusinessActivitie name="Costumer_Verification">
  <FormalParameters>
    <FormalParameter Mode ="IN">
      <DataType>
        <BasicType>STRING</BasicType>
      </DataType>
    </FormalParameter>
    <FormalParameter Mode ="OUT">
      <DataType>
        <BasicType>BOOLEAN</BasicType>
      </DataType>
    </FormalParameter>
  <FormalParameters>
  <ExecutionType> Manual </ExecutionType>
</BusinessActivitie>
```

(d)

Figure 4 One rule of the RbBPDL purchase order process (continued) (see online version for colours)

```
<!-- ..... Rule1 ..... -->
<Business_Rule Name= "R1" Version="1.0">
  <OnEvents> $Recieve_Order </OnEvents>
  <PreConditions> $Info_Costumer != "" </PreConditions>
  <Action>
    <Execute>
      <Operation>$Costumer_Verification </Operation>
      <InputVariableName>$Info_Costumer</InputVariableName>
      <OutputVariableName>$CostumerRegistred </OutputVariableName>
      <Performer>$Commercial_Service</Performer>
    </Execute>
  </Action>
  <PostConditions>$Database_Connexion = true</PostConditions>
  <PostEvents> $End_Costumer_Verification </PostEvents>
</Business_Rule>
```

(e)

In the next section, we detail our approach to manage the change of rules and estimate the impact and cost of this change on a business process.

5 Change management

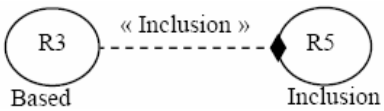
We aim at automating the management of changes that affect the flexibility of a business process by estimating the impact and cost of these changes. This should help in planning, organising, and managing the necessary resources that would carry out the changes. This estimation is beneficial when change alternatives are offered.

Our approach is as follows: firstly, we study the relationship between the rules to determinate the rules that are affected by a change in a certain rule. Secondly, we formalise the change management of a process by translating this later into a graph of rules. Thirdly, we determine the impact of a rule change by using an associated algorithm. Finally, we estimate the overall change cost by using the R2CM that takes into account two parameters: the nature of the relationships between rules and the rule distance in the graph of rules.

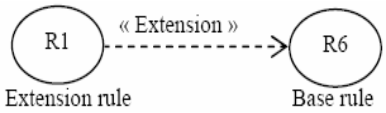
5.1 Relationships between rules

A change in a process element may require changing other elements that are related to this element for the sake of process consistency. Therefore, we need to study the relationships between the rules. We identify three relationships between rules:

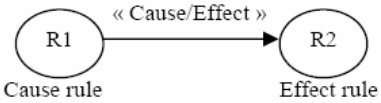
1 *Inclusion relationship*: shows the case of a rule (base rule) that includes the functionality of another rule (inclusion rule). Two rules have an inclusion relationship between them if the completion of the base rule's action requires the completion of the inclusion rule's action. In the previous example, to calculate the final price, the shipping price must be calculated before.



2 *Extension relationship*: shows the case of a rule (extension rule) that extends the functionality of another rule (base rule). Two rules have an extension relationship between them if the completion of the extension rule's action achieves the completion of the base rule's action. In the previous example, if we suppose that a loyal custom may receives a new discount. As a result, there is an extension relationship between R_1 (rule to identify a costumer) and R_6 (rule to calculate bill) because the functioning of R_1's action will complete the functioning of R_6's action.



3 *Cause/effect relationship*: shows the case of a rule (cause rule) that activates another rule (effect rule). Two rules have a cause and effect relationship between them if the execution of a rule will activate the effect rule. As a result, the execution of a cause rule's action triggers a post-event, which necessarily activates the effect rule. Thanks to this relationship, the order of process activities can be defined by describing the post-events based on ECAPE. In our previous example, the performance of R_1's action (verify costumer) will trigger end-customer – verification post-event. This latter is the event activator of rule R_2. There is a cause and effect relationship between R_1 and R_2.



Note that there is a slim difference between extension and cause/effect relationships. The extension relationship concerns the complementarity between rules without necessarily having an extension rule that activates a base rule. However, the cause/effect relationship concerns the activation rule without necessarily having a functioning complementarily between the cause and base rules. Another point is that the inclusion and extension relationships are manually defined by a designer, while cause/effect

relationship can be detected automatically by analysing the events and post-event parts in rules.

The fact of defining relationships between rules allows determining which rules must be revised in case of change. Firstly, all base rules which have an inclusion relationship with a changed inclusion rule must be revised by a business process designer. In the previous example, if the enterprise decides not to deliver its product rule R_3 will be deleted from the process model. The suppression of an inclusion rule (R_3) will affect a base rule, which requires the completion of the inclusion rule's action. Due to this, human intervention is required to decide how we can change a base rule in order to keep process coherence. Secondly, all base rules which have an extension relationship must be revised when an extension rule is changed. In the previous example, if we change rule R_1 (rule responsible for costumer identification), which represents an extension rule, then base rule R_6 (rule responsible for bill calculation) must be revised. Finally, all effect rules, which have a cause/effect relationship, must be revised if the cause rule is changed in order to ensure the activation of these rules. For example, the consequence of removing rule R_1 in the previous running example is the inactivation of R_2, because R_1 is the cause of activating R_2. For this purpose, a designer must revise the effect rules if the cause rule is changed.

5.2 Development of graph of rules

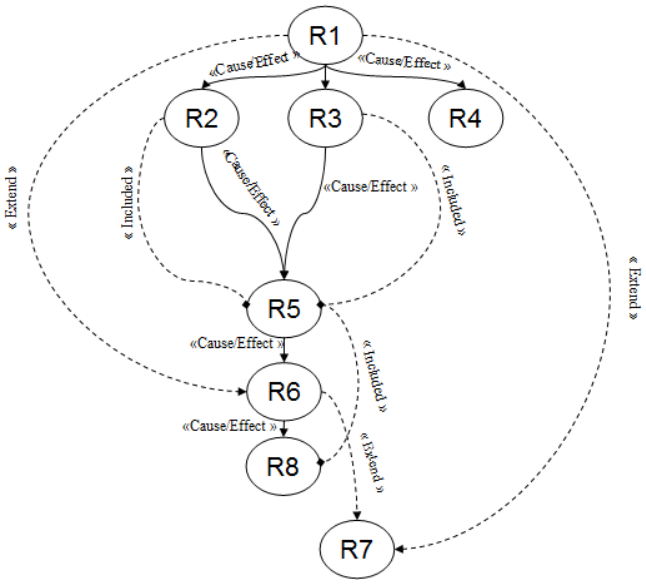
To formalise the flexibility management of a process model, we propose to translate a business process into a graph of rules. Vertices of this graph represent the rules, which are the business process, and arcs represent the relationships between the various rules. Three types of arcs are identified: include arcs that correspond to inclusion relationship; extend arcs that correspond to extension relationship, and cause/effect arcs that correspond to cause/effect relationship between rules. A graph of rules is formally defined as follows:

Definition 1: A graph of rules is a directed graph G_r (R, Y) with:

- R is a set of vertices that represent rules.
- Y is a set of arcs that represent three kinds of relationships.
  1. Y_i is a subset of Y such that if y_i (r_i, r_j) then r_i is included in r_j.
  2. Y_e is a subset of Y such that if y_e (r_i, r_j) then r_i extends r_j.
  3. Y_c is a subset of Y such that if y_c (r_i, r_j) then r_i cause the activation of r_j.

The rule graph of our previous example is illustrated by Figure 5. An inclusion arc is represented by a dashed arrow with a small diamond head on the side of the base rule. An extension arc is represented by a dashed arrow. Finally, a cause/effect arc is represented by a plain arrow. Note that two vertices can be linked by two arcs. For instance, R_3 is linked with R_5 by cause arc and extend arc (because R_3 cause the activation of R_5 and in the same time R_2 extend R_5).

Figure 5 Rules graph of the purchase order process



5.3 Change impact assessment

The graph of rules helps determine which rules are impacted by a change in a rule. If any vertex changes, all *direct successor vertices* must be revised. Formally this is defined as follows:

Definition 2: let G_r (R, Y) be a rule graph and r_i a vertex rule such that $r_i \in R$.

The set of r_i direct successor neighbours is noted as $N^+(r_i)$ such that $\forall r_j \in N^+(r_i)$, r_i is either inclusion, extension, or, cause rule for the base or effect rule r_j.

- We note $N_i^+(r_i)$ the set of direct r_i successors such that $\forall r_j \in N^+(r_i)$, r_i is an inclusion rule, for the base rule r_j.
- We note $N_e^+(r_i)$ the set of direct r_i successors such that $\forall r_j \in N^+(r_i)$, r_i is an extension rule for the base rule r_j.
- We note $N_c^+(r_i)$ the set of direct r_i successors such that $\forall r_j \in N^+(r_i)$, r_i is a cause rule for the effect rule r_j.
- We note $N_c(r_i)$ the set of direct r_i predecessors such that $\forall r_j \in N^-(r_i)$, r_j is a cause rule for the effect rule r_i.
- We note $N^*(r_i)$ the set of direct r_i neighbours such that $N^*(r_i) = N_i^+(r_i) \cup N_e^+(r_i) \cup N_c^+(r_i) \cup N_c^-(r_i)$. If $r_i \in R$ changes, then the designer will have to revise all rules $N^*(r_i)$.

To keep a process coherent, the change management of the process modelling will request from a designer to revise the $N^*(r_i)$ set when a rule r_i is changed. In the example of Figure 5, rule R_5 must be revised if rule R_3 is deleted because $N^*(R_4) = \{R_1, R_5\}$. The change management notifies the designer to revise rules R_1 and R_5 in order to decide how these rules can be changed. Note that we must

check the direct predecessor neighbours $N_c^-(r_i)$ for the cause/effect relationship since it is not acceptable that a rule activates a non-existing rule. For instance, if rule R_3 is deleted we will also have to revise rule R_1 to ensure that this letter does not activate a deleted rule.

However, when changing the set of direct successor neighbour's inclusion and extension rules ($N_e^+(r_i) \cup N_c^+(r_i)$) the designer should revise entirely the concerned rules. This revision may generate a cascade of rule change. Indeed, if one rule changes, the set of inclusion and extension rules will be revised and properly changed. This will raise the need to revise another set of successor neighbour's rules of the rule that was revised. In the example, if rule R_3 is changed, then rule R_5 (cause rule and inclusion rule) will be revised. This revision consists of analysing the entire code of rule R_5 to decide how we can change the latter in order to keep the coherence of the process. If we change rule R_5 after its revision, this results in revising rule R_6. In turn, rule R_6 can be changed after revision, this results into revising rules R_8 and R_7. And so on, until there are no rules to revise.

In contrast, to change the set of direct successor neighbour's cause rules ($N_c^+(r_i) \cup N_c^-(r_i)$) that do not generate a cascade of the change because the designer, in this case, the designer should only revise the event and post-event parts of the rules that are concerned. In the example, if we change rule R_3, then rule R_1 will be revised. This revision consists of updating the post-event to ensure that this letter does not activate a deleted rule (as we explained above). After this update, we do not need to revise another set of direct successor neighbour's rules.

The following algorithm summarises the change impact of a rule.

ChangeImpact_Procedure (R_x, stack S)
{ **if** NotExist(S, R_X) **then** // test if the rule's stack S contains the rule R_X
{ push (S, R_X); // push the rule R_X onto stack S }
if NotExist(S, $N_c^-(R_X)$) **then** R_X
{ push (S, $N_c^-(R_X)$); }
if NotExist(S, $N_c^+(R_X)$) **then**
{ push (S, $N_c^+(R_X)$); }
if $N_i^+(R_X) \neq \Phi$ **then**
{ ChangeImpact_Procedure ($N_i^+(R_X)$,S);
} **Else**
{ if $N_e^+(R_X) \neq \Phi$ **then**
{ChangeImpact_Procedure ($N_e^+(R_X)$,S);
}
Else
{ exit ();}}
}

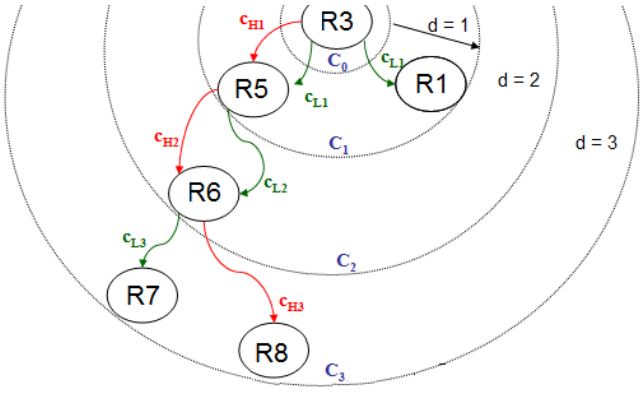
It should be noted that a change cascade is not a consequence of the change management that we propose. Indeed, this management is not about implementing changes but about guaranteeing process consistency. In the previous process, rule R_3's change cascade (rules R_1, R_5, R_6, R_7, and R_8) needs to be revised in order to ensure the activation of all the rules and the business coherence of the process as well. In the following, we suggest how a designer is given the possibility of assessing the efforts to put into per change.

5.4 R2CM model

In order to offer a tangible estimation of the efforts needed to implement rule changes, R2CM is used. Change cost is the necessary effort to modify the rules that are subject to changes following a change in a rule. For example, if rule R_3 is deleted and rule R_5 is changed in the previous example, so the effective change effort applicable to rule R_3 concerns the efforts to change rule R_1 plus the efforts to change rule R_5. However, it is more beneficial to estimate the maximum change cost before making any changes. This will indicate to a designer the cost of a planned change. For this reason, in the R2CM the term 'cost of change', denoted by $\zeta(R_i)$, is used to designate the maximum change cost before a change occurs.

The R2CM is based upon a rule change impact graph which is derived from the graph of rules (Figure 6). The new graph is defined where vertices represent rules, arcs represent relationships between the various rules, and there exists one vertex that represents the changed rule and does not have predecessors. Note that, the predecessor neighbours cause rules of the changed rule in the graph of rules become the successor neighbours cause rules in the change impact graph because these latter are impacted by the rule change (in the previous example, rule R_1 become a cause rule successor on the changed rule R_3).

Figure 6 the change impact graph of the rule R_3 (see online version for colours)



By using a rule change impact graph, the R2CM computes the cost of change as a function of two parameters:

1 the distance between a changed rule and each impacted rule in this graph

2 the nature of the relationships between a changed rule and each affected rule in this graph.

Firstly, the overall cost of change of rule R_i is the sum of the cost change of the rules with different distances in a rule change impact graph. Formally, this will be defined as follows:

$$\zeta(R_i) = \sum_{i=0}^{d_{max}} C_i \quad (1)$$

where C_0 represents the cost of change of rule R_i. In the previous example $\zeta(R_3) = C_0 + C_1 + C_2 + C_3$.

However, according to Xiao et al. (2007), the nodes with the shortest distance are more likely to be directly impacted by changes than the node with the longest distance away from the changed node. Consequently, the change cost of the rules with the shortest distance is greater than the change cost of the rules with the longest distance. Formally,

$$C_i = \alpha C_{i-1} \quad (2)$$

where α is a constant which is always between zero and one ($0 < \alpha < 1$). In this way, from formulas (1) and (2) we deduct that the overall cost of change $\zeta(R_i)$ is a geometrical series with α as a constant ratio. The general term of this series is given as follows:

$$\zeta(R_i) = C_0 \frac{1-\alpha^{d_{max}}}{1-\alpha} \quad (3)$$

Secondly, according to the nature of the relationships between rules, two qualifications for the change cost can be considered:

- We qualify *high* change cost (C_H) the effort to put into changing inclusion and extension relationships because the designer has to revise entirely the rules concerned.
- We qualify *low* change cost (C_L) the effort to put into changing a cause/effect relationship because the designer has to revise the event and post-event part of the rules concerned.

In this way, the rules cost change with distance i denoted by C_i is defined as follows:

$$C_i = n_i c_{L_i} + m_i c_{H_i} \quad (4)$$

where n_i is the number of the case rules at distance i and m_i is the number of inclusion and extension rules at distance i. However, as explained above, the change cost of these rules is higher then change cost of cause/effect rules. Consequently, formula (4) becomes:

$$C_i = (n_i \beta + m_i) c_{H_i} \quad (5)$$

where β is a constant which is always between zero and one ($0 < \beta < 1$).

To sum up, the *R2CM* estimates the effort needed to implement the rule change by using the following formulas:

$$\begin{cases} \zeta(R_i) = C_0 \dfrac{1-\alpha^{d_{max}}}{1-\alpha}, \text{ such that } \alpha \in]0,1[\\ \forall i \in [1, d_{max}], C_i = (n_i \beta + m_i) c_{H_i}, \text{ such that } \beta \in]0,1[\end{cases}$$

To demonstrate the feasibility of the R2CM and to determine α and β values, a series of experiments can be conducted by using a set of real business processes and studying the impact of changes on processes. The purpose is to elaborate a consistent mathematical model of α and β (or interval).

6 Process verification

Companies must have reliable business processes to achieve their objectives. Reliability is a crucial issue because it lets companies capitalise on their information systems. A formal verification of a process is required to ensure that this process meets all the agreed-upon requirements. In our rule-based approach, we use formal models to identify possible functional errors. Among the different models used in the literature for this type of verification, we opted for PNs (Van der Aalst, 1998). In this section, we detail how to write an ECAPE rule using PNs. However, among the various types of PNs, which one does support the semantics of ECAPE rules set?

6.1 PN for business process verification

PN was introduced to model and analyse the behaviour of systems based on a graph model. A PN is a bipartite directed graph with two types of nodes: a *place* models a condition or state of a system resource, and a *transition* models an event or action that takes place within the system. The conditions necessary to trigger an action are modelled by arcs that connect one place to one transition. For modelling the dynamic aspect of the system, *tokens* are used. Indeed, if a place contains a token that means the condition represented by this place is verified or indicates the availability of a resource in case several tokens exist in one place. One transition is *fired* if there are a defined number of tokens at the input place. After firing a transition, the tokens are consumed from input places and will be placed at the end of all output places. Finally, a PN *marking* is the distribution of tokens over the places at a given instance.

To detect errors that a business process could contain, several studies have exploited the strengths of PN. An example of this is the works conducted by research team of Eindhoven University of Technology, which proposes to use a PN, called workflow nets (WF-nets), to validate and verify workflows (Van der Aalst, 1998). WF-nets require a single initial and final place, transitions represent real activities, or routing and places represent pres and post-conditions. However, the regular PNs miss expressiveness since they cannot distinguish the existence of two tokens in the same place. For this reason, several PN extensions have been proposed. The well-known extension is the coloured Petri net (CPN). Indeed, in this kind of net, tokens are typed (coloured) to express the different characteristics of resources. In this way, the pre-conditions such as "only tokens which have of a given colour can be are fired" can be supposed. The CPN is widely used to verify web services composition and business process. An example of this is the paper of Yang et al. (2005).

Unfortunately, the semantics of the classical PN and CPN do not allow modelling the behaviour of ECA rule-based process (specifically, the ECAPE rule-based process). In classical Petri nets (or CPNs), when all the conditions necessary for firing one transition are satisfied, the transition may be fired but not necessarily, while in an ECA system, it must be fired. According to Eshuis and Dehnert (2003) the environment of classical Petri net models closed systems because it does not influence the firing of transitions. In contrast, an ECA system is open because it usually needs some additional input event to become enabled. For this reason, Eshuis and Dehnert (2003) propose in a reactive Petri net (RPN) which considers two type of transitions: internal transitions ($T_{internal}$) and external transitions ($T_{external}$). The firing rule of $T_{internal}$ is 'the transition must be fired' rather then 'the transition can be fired'. At the same time, the external transitions interaction with the environment and the classical firing rule can is preserved to ensure the stability of the PN. Therefore, RPN has two possible states: stable and unstable. A state is stable if no internal transition is enabled, it is unstable otherwise. A stable state can become unstable if some external transition fires. In an unstable state, the RPN must fire some enabled internal transitions. By firing these transitions, a new state is reached. If the new state is stable, the PN has finished its reaction.

Despite possibility of using RPNs to model ECAPE rules set, this Net is not enough. It does not explicitly include the necessary constructions to model complex events. We are interested in another PN called conditional coloured Petri net (CCPN). This CPN is proposed by Li and Marín (2004) in order to model an ECA rule for an active database system. The particularity of this PN is the fact of defining some new elements on CPN to characterise ECA rules features. Especially, the complex events can be modelled by defining new types of places and new types of transitions. As a result, ECA rules can be easily modelled by CCPN by considering the events and actions of one rule as places which are inputs and outputs transition, ECA rules themselves are mapped onto transitions and conditions are attached to transitions. However, a CCPN cannot model perfectly an ECAPE rules set, because each ECAPE rule may trigger one or more rules by describing explicitly the events triggered. As a result, the rule sequence can be automatically deducted.

Taking into account how CCPN models complex events, we suggest a new CPN called ECAPE net. This latter allows rewriting formally an ECAPE rules set in order to analyse and verify the business processes

6.2 ECAPE net

An ECAPE net is a CPN that models the ECAPE rules execution sequence, which represents the logic of a business process. A rule's events are represented by places. The input places represent events that activate one rule and the output places represent the events triggered by the execution of a rule action. In turn, a rule's action is represented by transitions. Finally, a rule's condition (and a rule's post-condition) is attached to a transition (Figure 7). However, in an ECAPE rule, an event can be primitive or composite. In addition, a rule's action consists of various predefined instructions.

Figure 7 General structure of ECAPE net

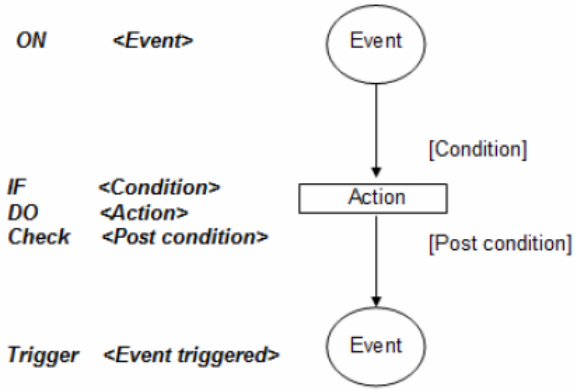
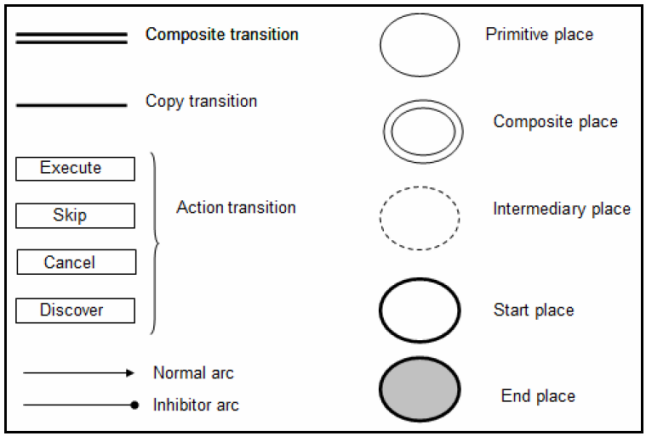


Figure 8 ECAPE net elements



For this reason, we need new PN elements that would help build composite events from primitive events and to model an action's instructions. Figure 8 illustrates the ECAPE net's elements. In this PN, there are five types of places, two types of transitions, and two types of arcs:

- *Primitive place* used to represent a primitive event.
- *Composite place* used to represent a composite event.
- *Intermediary place* used specially in ANY event constructor to build a composite event.
- *Start place* used to represent the first primitive event that launches the process execution.
- *End place* used to represent the last primitive event that causes the end of process execution.
- *Composite transition* used to generate a composite event from primitive event. This will be explained later.
- *Action transition* used to represent the instructions of a rule's action.
- *Normal arc* used to represent the PN control flow.

- *Inhibitor arc* used specially in NOT event constructor to build negation composite events.

Now, we give a formal definition of an ECAPE net.

Definition 3: An ECAPE net is a CPN with 11-tuple ECAPE net = (Σ, P, T, A, C, W, Activity, Cond, PstCond, D, τ):

- Σ is a finite set of types also called colour sets.
- *P* is a finite set of places that model the rule's event. P is divided into three subsets: $P = P_{prim} \cup P_{comp} \cup P_{inter}$ where P_{prim}, P_{comp} and P_{inter} are set of primitive, composite and intermediary places. P_{prim} contains two subsets: $P_{start} \in P_{prim}$ and $P_{End} \in P_{prim}$ where P_{start} and P_{End} are the set of start place and end place.
- *T* is a finite set of transitions. T is divided into two subsets: $T = T_{comp} \cup T_{copy} \cup T_{action}$ where T_{comp} and T_{action} are a set of composite and action transitions. T_{action} is divided into four subsets: $T_{action} = T_{Execute} \cup T_{Skip} \cup T_{Cancel} \cup T_{Discover}$ where $T_{Execute}$, T_{Skip}, T_{Cancel} and $T_{Discover}$ are the set of Execute, Skip, Cancel, and Discover transitions.
- *A* is a set of directed arcs that connects a place to a transition and vice-versa such that $A \subseteq (P \times T) \cup (T \times P)$. A is divided into two subsets: $A = A_{norm} \cup A_{inhi}$ where A_{norm} and A_{inhi} are set of normal and inhibitor arcs.
- *C* is a colour function that assigns a unique colour to each place p. The colour of a place is denoted by C(p). C is defined from P to Σ such that: $\forall p \in P, \exists C(p): C(p) \in \Sigma$
- *W* is a function that defines arcs by determining the token's variables that are either consumed or produced during operation such that $\forall a \in A$, $Type(var(W(a)) = C(p(a)))$ and $Type(var(W(a))) \subseteq \Sigma$ where Type(var(W(a))) is a function that determines the types of the variables in an arc. In this formula, the first part, expresses that the types of the arc's variables must be compatible with the colours set of the input place or output place. And, the second part expresses that the token's types must belong to the colours set of the CPN.
- *Activity* is a function that is defined from T_{action} to an activity name.
- *Cond* is a condition function that is defined from either T_{action} or T_{comp} to expressions such that :

  $\forall t \in T_{action}, Type(Cond(t)) = Boolean$ where Cond function evaluates the rule's condition;

  $\forall t \in T_{comp}, Type(Cond(t)) = Boolean$ where Cond function evaluates the condition of a composite transition.

- *PstCond* is a condition function that is defined from T_{action} to expressions such that :

  $\forall t \in T_{action}, Type(Cond(t)) = Boolean$ where PstCond function evaluates the rule's post condition.

- D is a time interval function that is defined from T_{comp} to a time interval $[d_1, d_2]$
- τ is a time stamp function that assigns each token in place p a time stamp corresponding to time event happen. t is expressed in natural clock with the form year: month: day – hour: minute: second. For example, a token has time stamp 2009: 02: 06 –18: 46: 16.

Note that, in ECAPE net, a token is four-tuple (p, c, data, timestamp) where $p \in P$, $c \in C(p)$ called the colour, data is business information and timestamp specifies the natural time when the token is placed into place p.

Before continuing to explain the ECAPE net, we will first, define some usual functions, sets, and notions.

Firstly, M is the net marking function that is defined from P to N (set of all natural numbers) to assign to each place a number of tokens. We denote by M_0, the initial marking (initial state) of the PN. And we denote by M_f, the final marking (final state) of the ECAPE net. M(p) is the place marking function that is defined from P to N (set of all natural numbers) to specify the number of tokens in one place at a given state.

The state of the net may change according to the number of tokens during the execution of the net. Indeed, A state M_n is called reachable from M_1 (notation $M_1 \xrightarrow{*} M_n$) if and only if there is a firing sequence $\sigma = t_1 t_2 ... t_{n-1}$ such that $M_1 \xrightarrow{\sigma} M_n$.

Secondly, we define the five following input/output sets:

- ${}^{\bullet}t$ is the input places set of transition t such that ${}^{\bullet}t = \{p \in P : (p,t) \in A\}$.
- ${}^{\bullet}_{n}t$ is the input normal arc places set of transition t where the arc that connects p to t is a normal arc such that ${}^{\bullet}_{n}t = \{p \in P : (p,t) \in A_{normal}\}$.
- ${}^{\bullet}_{i}t$ is the input inhibitor arc places set of transition t where the arc that connects p to t is an inhibitor arc such that ${}^{\bullet}_{i}t = \{p \in P : (p,t) \in A_{inhabitor}\}$.
- $t^{\bullet}$ is the output places set of transition t such that $t^{\bullet} = \{p \in P : (t,p) \in A\}$.
- $p^{\bullet}$ is the output transitions set place p such that $p^{\bullet} = (t \in T : (p,t) \in A)$.

Finally, in an ECAPE net, a sequence S from a node n_1 (place or transition) to a node n_k is a sequence $<n_1, n_2, ..., n_k>$ such that $<n_i, n_{i+1}> \in A$. A sequence is elementary if each node is unique.

An ECAPE net models the execution of ECAPE rules set. To this end, we specify the firing rules of composite and action transition of an ECAPE net.

6.2.1 Definition of the composite transition firing rule

A composite transition is fired if there is at least one token in all its input places, the sum of all tokens in all input places is more then the multiplicity of normal arcs that connect all input places to this composite transition, and all conditions attached to the composite transition is true. Formally, a composite transition is fired if and only if:

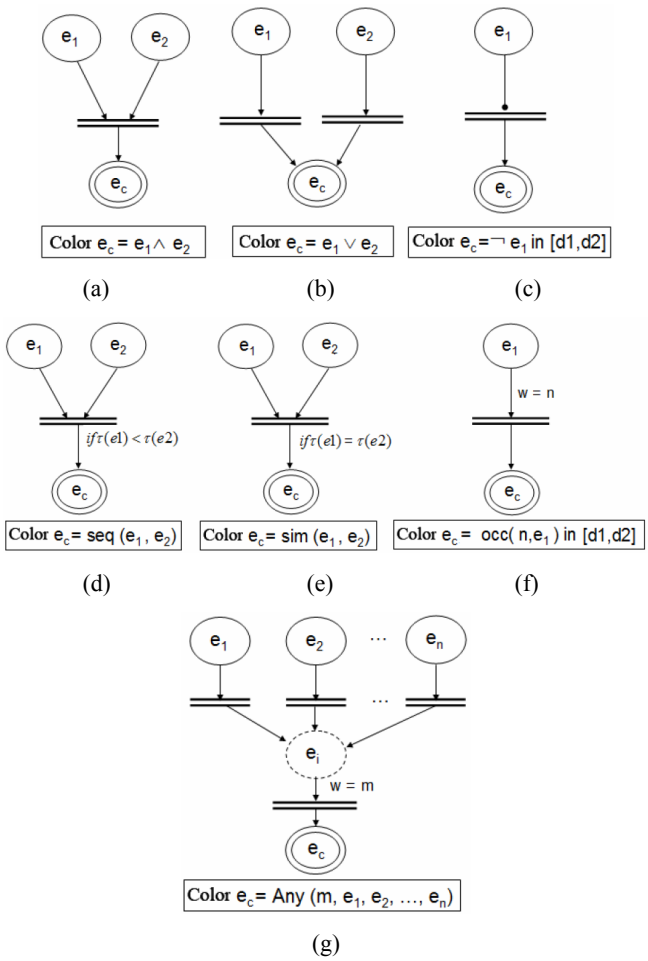
1 $\forall p \in {}^{\bullet}_{n} t_{comp}, M(p) \geq 1$

2 $\forall p \in P, M(p) \geq W(p, t_{comp})$

3 $Cond(t_{comp}) = True.$

Note that a composite transition is also fired when there is no token in the time interval $[d_1, d_2]$ at the input places that are connected with inhibitor arcs.

Figure 9 ECAPE net structures of composite events



Formally, a composite transition is fired if and only if: $\forall d \in [d_1, d_2], \neg\exists p \in {}^{\bullet}_{i} t_{comp}, M(p) \geq 1$. As stated before, a composite transition is used to generate a composite event:

- Conjunction event ($e_1 \wedge e_2$) expresses that both e_1 and e_2 happen. Figure 9(a) shows how to generate this event. The composite transition is fired when both e_1 and e_2 are marked. After firing this transition, the composite place e_c is marked.
- Disjunction event ($e_1 \vee e_2$) expresses that e_1 or e_2 happen. Figure 9(b) shows how to generate this event. In fact, one of the two composite transitions is fired when e_1 or e_1 are marked. After firing one of the two transitions, the composite place e_c is marked.
- Negation event ($\neg e_1 in[d_1, d_2]$) expresses that e1 does not happen in the time interval $[d_1, d_2]$. Figure 9(c) shows how to generate this event. In fact, the composite transition is fired when e_1 is not marked in the time interval $[d_1, d_2]$. After firing this transition, the composite place e_c is marked.
- Sequence event ($SEQ(e_1, e_2)$) expresses that event e_2 before event e_1. Figure 9(d) shows how to generate this event. In fact, the composite transition is fired when both e_1 and e_2 are marked and if $\tau(e_1) < \tau(e_2)$. After firing of this transition, the composite place e_c is marked.
- Simultaneous event ($SIM(e_1, e_2)$) expresses that event e_1 happens at the same time with event e_2. Figure 9(e) shows how to generate this event. In fact, the composite transition is fired when both e_1 and e_2 are marked and if $\tau(e_1) = \tau(e_2)$. After firing of this transition, the composite place e_c is marked.
- Occurrence event ($OCC(n, e_1)$ in $[d_1, d_2]$) expresses that event e_1 occurs n times in the time interval $[d_1, d_2]$. Figure 9(f) shows how to generate this event. In fact, a composite transition is fired when e_1 marked with n token in the time interval $[d_1, d_2]$. After firing this transition, the composite place ec is marked.
- Any event ($ANY(m, e_1, e_2, \ldots, e_n)$) expresses that any m events in $e_1, e_2, \ldots, e_n$ happen where $m < n$. Figure 9(g) shows how to generate this event. In fact, m composite transitions are fired when $e_1, e_2, \ldots, e_n$ are marked. Each of these m transitions mark an intermediary place e_i. This will help limit the composite transitions fired to m because when e_i has m token, the last composite transition is fired to generate the composite event e_c.

6.2.2 Definition of the action transition firing rule

An action transition is used to represent the instructions of a rule's action. It is fired if there is at least one token in all its input places and all conditions and post-conditions attached to it is true. Formally, an action transition is fired if and only if:

1 $\forall p \in {}^{\bullet} t_{action}, M(p) \geq 1$

2 $Cond(t_{action}) = PstCond(t_{action}) = True.$

However, the skip transition (T_{Skip}) and cancel transition (T_{Cancel}) have some particular semantic execution.

As shown in Figure 10, the cancel transition is attached to an Execute transition. If cancel transition is fired, all tokens of all input places of its attached execute transition are removed. And, cancel transition disables execute transition. The skip transition is also attached to execute transition. If skip transition is fired, all tokens of all input places of its attached execute transition are removed and one token is added into all output places of it is execute transition. Skip transition skips execute transition.

Figure 10 Cancel and skip transition pattern

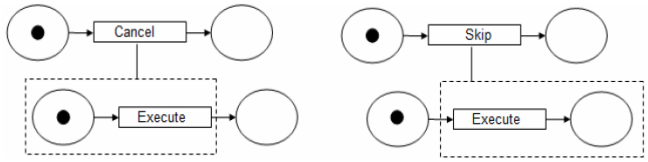


Figure 11 ECAPE net of the purchase order

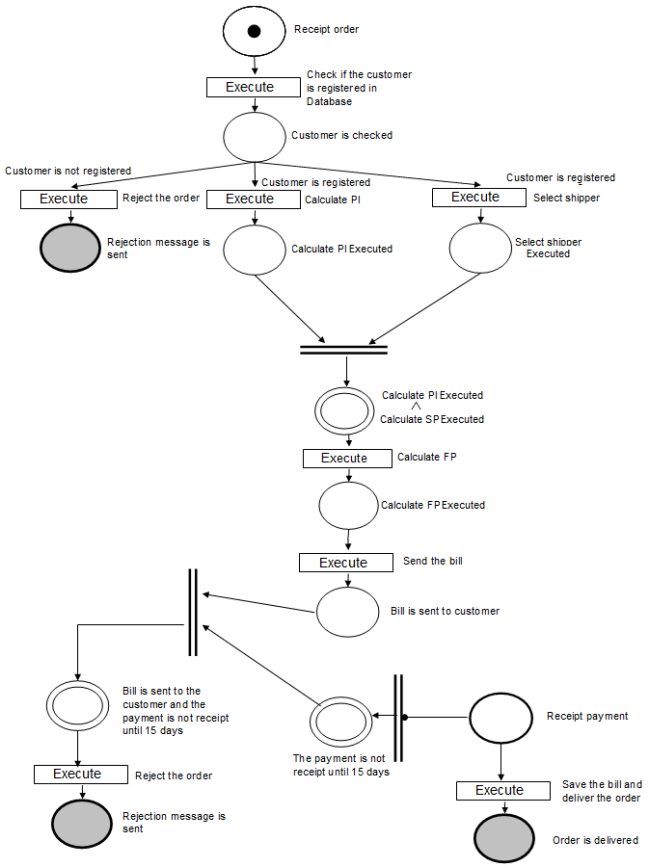


Figure 11 illustrates an ECAPE net of the purchase order process example. Here, the events are represented with places. The composite events are generated using composite transitions and represented by composite events. A rule's actions are represented with action transitions. Finally, the rule's conditions are attached to action transitions. We detail the semantics of this PN execution through the following execution scenario example. We assume that the first place has one token when event (receive order) of rule R_1 occurs. So this place is a start place because it launches the execution of the process. After that marking, the action transition, which represents the execution of the action request order of rule R_1, is fired. Consequently, one token is placed in the place, which represents post-event of rule R_1 and triggered by executing R_1's action (check if the customer is registered in database).

After that, if the condition 'costumer is registered' is satisfied then the two action transitions, which represent respectively the initial price calculation action of rule R_2 and shipper selection action of rule R_3, are fired. Due to this firing, one token is placed in each output place of these two action transitions. However, to calculate the final price, we need to complete the execution of the initial price calculation action and shipping price calculation action. To model that, we use a composite event that is generated by a composite transition as shown in the figure. This transition is fired only if one token is placed in each input places. According to our execution scenario, the composite transition is fired after firing the action transition, which represents the shipping price calculation action of rule R_3 and after placing one token in the output primitive place because one token is already placed in a place that shows that the initial place calculation action is completed. And so on, the token moves along the primitive places, composite transitions, composite places and action transitions until it reaches n end place, which represents the end process event.

After detailing the formal ECAPE net definition, we will show how to verify the business process using this net.

6.3 Verification of process using ECAPE net

Verification ensures the correctness of a process. According to Van der Aalst (1998), a business process definition is correct if a set of minimal requirements or criteria is satisfied. These criteria can be related to the structure of the PN (so-called *well-structuredness property*) or to the dynamic of the PN itself (so-called *soundness property*).

6.3.1 Well-structuredness property

The well-structuredness property is proposed in Van der Aalst (1998) to formalise the need to satisfy that, in a WF-net, every split (OR, AND, etc.) is followed by a corresponding join of the same type. This property can be used to verify the structure of the UML activity diagrams and BPEL4WS code (Dehnert and Zimmermann, 2005). However, in an ECAPE rule, the control flow is implicitly defined and each rule has a condition and post-condition. For this reason, in terms of ECAPE net, the well-structuredness property is characterised by the need to satisfy, in addition to structural properties, the satisfiability of the predicates that represent the condition and the post-condition of rules. An ECAPE net is well-structuredness if the following three properties are verified:

- supposes that:
  1 an ECAPE net has at least one start place p_{start} and at least one end place p_{end}

2 each node of ECAPE net is on a sequence from the P_{start} place set to P_{end} place set.

Formally, the regular property requires to satisfy two conditions:

1 $P_{start} \neq P_{end} \neq \emptyset$

2 $\forall n \in P \cup T \; \exists S$ such that $n \in S \wedge p_{start} \in S \wedge p_{end} \in C$.

- *Completeness property* supposes that each place (or transition) of ECAPE net is linked to, at least, one transition (or place) except the start place and end place. Formally, the completeness property requires to satisfy two conditions:

  1 $\forall p \in P - P_{end}, \exists t \in T$ such that $t \in p^{\bullet}$

  2 $\forall t \in T, \exists p \in P - P_{start}$ such that $p \in t^{\bullet}$.

- *Satisfiability property* supposes that:

  1 each condition (post-condition) of one transition is satisfied

  2 for each ECAPE net path the conjunction the condition and post-condition of transitions $t_1, t_2 \ldots t_{j-1}$ forming this path is satisfied.

  Formally, the satisfiability property requires to respect the following conditions:

  1 $\forall t \in T$, Cond(t) is satisfied and PstCond(t) is satisfied

  2 $\forall s \in$ Sequences, $\vdash \underset{t \in C}{\wedge} \text{Cond}(t) \wedge \text{PstCond}(t)$ is satisfied such that Sequences is the set of distinguish sequence of an ECAPE net.

The ECAPE net of the previous example (Figure 14) is Well-structured because it is regular, complete, and each ECAPE net sequence, the conjunction of the condition and post-condition of transitions forming this path is satisfied $\vdash$ (costumer is registered) $\wedge$ (IP + SP $\leq$ FP) $\wedge$ ($\tau(e_1) < \tau(e_2)$) is satisfied, $\vdash$ (costumer is registered) $\wedge$ (IP + SP $\leq$ FP) is satisfied and $\vdash$ (costumer is not registered) is satisfied. For this reason, the satisfiability property is verified.

6.3.2 *Soundness property*

Soundness property has been introduced in Van der Aalst (1998) to formalise the need to satisfy that, in a well-structured PN, there are no dead transitions and neither Deadlock nor Livelock. A dead transition occurs when all transitions are reachable. A Deadlock occurs if a jam happens before the condition 'end' is reached. Finally, Livelock occurs if a subset of transitions are fired in an endless cycle. Furthermore, a well-structured PN is sound if termination is always possible and once terminated there is no residual tokens in the places. In term of well-structured ECAPE net, the soundness property is characterised by need to satisfy the following conditions:

- For every state M reachable from initial state, there exists a firing sequence leading from state M to end state. Formally:

$$\forall M\left(M_o \xrightarrow{*} M\right) \Rightarrow \left(M \xrightarrow{*} M_f\right)$$

- Final state is the only state reachable from initial state with at least one token in place p_{end}. Formally:

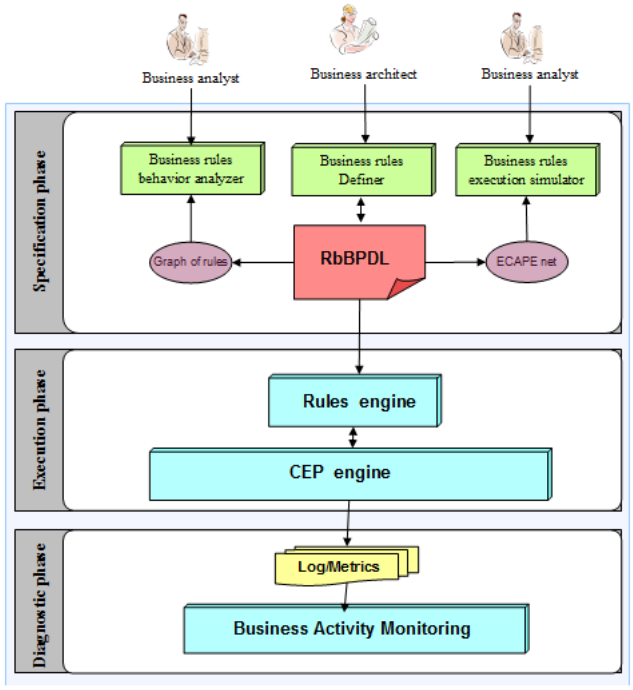
$$\forall M\left(M_0 \xrightarrow{*} M \wedge M \geq 0\right) \Rightarrow (M = 0)$$

- There are no dead transitions in (ECAPE net, p_{start}). Formally:

$$\forall t \in T \exists M, M' \; M_0 \xrightarrow{*} M \xrightarrow{t} M'$$

The ECAPE net of the previous example (Figure 12) is sound because, it is well-structured. Secondly, by simulating the net, we deduce that there are no dead transitions and neither Deadlock nor Livelock (termination is always possible). And once terminated there are no residual tokens in places.

Figure 12 The architecture of the BPFAMA framework (see online version for colours)



7 The BPFAMA architecture

In this section, we describe the architecture of the business process framework for agility of modelling and analysis (BPFAMA), which uses ECAPE rules to model business processes in a declarative way. BPFAMA addresses two issues: the implementation of rules in business processes' codes make these processes rigid and difficult to maintain, and the lack of mechanisms to support the verification of processes. The architecture of the BPFAMA is provided in Figure 12. The RbBPDL is a cornstone in BPFAMA by

adopting the ECAPE formalism to describe a business process using a set of rules

In the specification phase, designers define the elements of a new business process or redefine some elements of an existing process in order to improve it. The BPFAMA *rules definer* (which represents the business plane, see Appendix 2) uses RbBPDL to define business processes. One of the requirements on the rules definer is to provide a convivial representation and easier way to model business processes. A set of complete graphical notations (graphics and charts) is needed to represent graphically an RbBPDL process. We use URML notations that are proposed by Wagner et al. (2006) to describe the rules with graphical notations and meta-models inherited from its ancestor UML. As a result, the BPFAMA rules definer uses URML to graphically describe a business process and transforms the URML representation into an RbBPDL format.

After translating a process into a graph of rules and an ECAPE net, the BPFAMA *rules behaviour analyser* (which represents the behaviour plane, see Appendix 2) ensures the flexibility of this process by analysing its corresponding graph of rules. Afterwards, the *rules simulator* (which represents the operational plane, see Appendix 2) verifies the process functioning by analysing the ECAPE net of this process.

In the execution phase, a *rule engine* interprets the fulfilment specification of the actions reported in the rules. This interpretation is performed by automating the interactions between business participants (information and tasks) and allocating the different resources. The activation of these rules is ensured by a *complex event processing (CEP) engine*. This engine detects the predefined events (primitive of complex events) and alerts the Rule engine through messages. When the Rule engine receives a message, it activates the rules according to the triggered events. Note that, these messages are also submitted back to the CEP engine that will treat these messages as incoming events. This allows generating composite events. Finally, the different log files, historic of events, and traces compiled during the execution of rule instances over time are stored in a specific base to be used in the diagnostic phase.

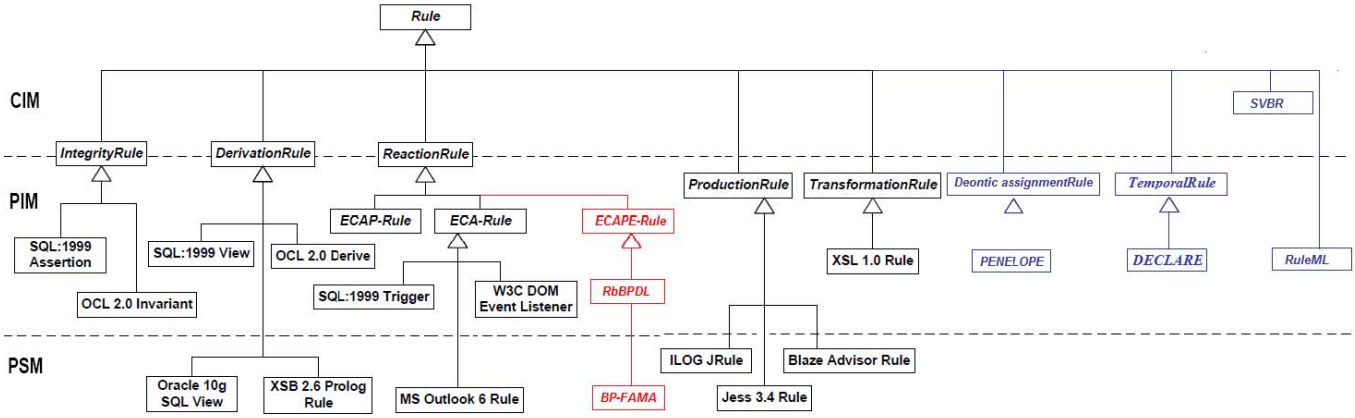
In the diagnostic phase, *a business activity monitoring (BAM)* is used to provide real-time information about the status and results of the various rules' actions, the various events triggered, and the transactions. Therefore, the enterprise is better informed and can make appropriate decisions.

8 Related work

Our main motivations stem out of the importance of improving BPM in terms of flexibility and verification. In this section, we discuss two major research directions: what is the rule formalism appropriate to manage changes in processes and what is the impact of these changes on processes?

First, the combination of business processes and rules has been studied for several years. There is a good number of research approaches that looked at the pros and cons of this combination. These approaches could be classified into two categories. The first approach considers that flexibility should be an important element of the imperative definition languages of processes. For this reason, the first category of works proposes to encapsulate rules with web services, like the work of Lee et al. (2003). Second category of works proposes to extend the imperative languages in order to take into account the rules in process model, like the work of Van Eijndhoven et al. (2008) which suggest to uses patterns to model the process parts that are most variable. And also, the work of Boukhebouze et al. (2007) which proposes to identify rules in a business process model by adding a 'rule' activity to BPEL.

Figure 13 Rule languages at different levels of abstraction (see online version for colours)



Source: Wagner (2005)

Despite the solutions provided by this approach which respond in some way to the problem of imperative languages rigidity, they do not guarantee complete flexibility including the impact of the change of a rule on the rest of the process. Moreover, experiments have shown that organisations express their policies and regulations in rules form using natural language or adding text annotations in their models (Zur Muehlen et al., 2007). However, the formulation of rules must be rigorous, concise and precise to ensure that these rules are unambiguous, coherent, and set out with a common business vocabulary.

In this spirit, a second approach proposes to model the logic of the process with a set of rules using rules languages According to Wagner (2005), the rules models can be classified, in accordance with to MDA architecture, as following three levels of abstraction (Figure 13).

8.1 Computation independent modelling (CIM)

Rules metamodels are proposed in the aim to define the vocabulary used to express the rules. Indeed, the business vocabulary definition can be textually using structured English, as proposed by the SBVR in OMG (2008). The business vocabulary definition can also be appeared visually as conceptual class diagrams in UML, as proposed by Wagner (2005). Finally, the business vocabulary definition can formally be described as predicate logic or as ontologies in RDF and OWL (Wagner, 2005).

8.2 Platform independent modelling (PIM)

Rigorous, concise and precise to ensure that these rules are unambiguous rules models, supported by languages, are proposed to formalise rule expressions. Indeed, the rules formalism used in these models depend to what categories of rule they represent. An example of theses languages, the PENELOPE (Goedertier and Vanthienen, 2006) that uses the Deontic logic to formalise the rules in terms of obligations and authorisations that feature business interactions. Another example is the DECLARE language (Pesic et al., 2007) that uses temporal logic to formalise the rules that control the execution order of the activities of a process. Note that, some general rule markup languages are proposed. These languages can be used for interchanging rules between different rule languages like RuleML (Schroeder and Wagner, 2002).

However, RuleML language is not appropriate for defining a process since elements such as business activities, participants, and event, just to cite a few are excluded. Furthermore, according to Knolmayer et al. (2000) and Lu and Sadiq (2007) the reaction rules (ECA) are the most adapted to model rules. This is done in various works, like the AgentWork framework of Müller et al. (2004), where ECA rules are used for temporal workflow management. Our work is positioned in this rule category.

Thus, the use of the ECA formalism is interesting to model, in our framework, business processes. However, in the aforementioned declarative process modelling languages used this formalism, the modelling flexibility with focus on the impact of a rule change on the rest of a process is not well looked into. Therefore, there is a need for a more powerful formalism that would allow a complete definition of this relationship. This way, we opted for the use of ECAPE formalism and RbBPDL language.

8.3 Platform specific modelling (PSM)

The execution rule models are proposed in order to formalise the execution of the rules set as ILOG JRules. However, these execution rule models do not allow having an explicit execution scenario. As a result, a more powerful paradigm is deemed appropriate in order to translate, in an easy way, a business process into a formal model and ensure the process verification allowing to building an execution scenario in an automatic way. This is why we opted for the use of ECAPE formalism and ECAPE net.

Secondly, how to analyse the impact of software change is a research topic for several years. This is why literature teems with proposals that attempt to answer this delicate question. Like OMEGA project proposed in Chen et al. (1996), which identify the propagation effects caused by code modification in C++ program. Note that, some cost models are proposed in order to estimate necessary effort to a software development. An example of this is the famous COCOMO model (Boehm, 1981). In parallel, some works were interested in analysing the impact of a business process changes. For instance, the work of Xiao et al. (2007) that proposes an approach to support impact analysis by using change impact metric. This metric is based on distance between a changed rule and each affected rules in a generated propagation graphs. This approach is some where, similar to our proposed approach. However, the different between Xiao's approach and your, is that we proposed to model a business process as a set of rules. This allow, in the hand, deploying partially-specified process definitions. In other hand, definition of this relationship between rules in other to manage the impact of change efficacy.

9 Conclusions

In this paper, we proposed a rule-based model to address the following two issues: the implementation of rules in business processes makes these processes rigid and difficult to maintain, and the lack of mechanisms to support the verification of these processes. Our model adopted the ECAPE formalism to describe processes with rules that are afterwards translated into a graph of rule and an ECAPE net to manage their flexibility and simulate their execution, respectively. The representation of the rules is done with an XML-based language called RbBPDL. Finally, we proposed the BPFAMA as an integration environment of the different elements we proposed. This environment consists of different tools namely: *rules definer*; *rules behaviour analyser* and *rules simulator*.

In term of future work, we aim to ensure the reliability of the business process execution by using a self-healing strategy. To this end, we will attempt to propose a

self-healing strategy for the process RbBPDL on the basis of the ECAPE format by trying to identify potential risks of exceptions and by lunching exception handling in parallel with the process execution to intercept the exceptions when it take place and react in order to drive the process execution towards a stable situation. Another future work consists of proposing a set of ECAPE rule patterns that allow describing recurrent problems and solutions proposed to model a business process using the ECAPE model. The objective is to support the re-use of ECAPE rules in distinct process definitions.

References

Boehm, B. (1981) *Software Engineering Economics*, Prentice Hall, ISBN: 0138221227.

Boukhebouze, M., Amghar, Y. and Benharkat, A.N. (2007) 'BPFAMA: business process framework for agility of modeling and analysis', in *ICEIS: 10th International Conference on Enterprise Information Systems ICEIS*, Barcelona, Spain.

Business Rules Group (BRG) (2002) *Defining Business Rules, What are they Really?*', available at http://www.businessrulesgroup.org (accessed on July 2000).

Chen, X., Tsai, W., Huang, H., Poonawala, M., Rayadurgam, S. and Wang, Y. (1996) 'Omega-an integrate environment for C++ program maintenance', in *Software Maintenance Proceedings*, November, pp.114–123, CA.

Dehnert, J. and Zimmermann, A. (2005) 'On the suitability of correctness criteria for business process models', *Proc. 3rd Int. Conf. on Business Process Management (BPM 2005)*, September, pp.386–391, Nancy, France.

Eshuis, R. and Dehnert, J. (2003) 'Reactive Petri nets for workflow modeling', in *Application and Theory of Petri Nets 2003*, pp.296–315, Springer.

Giurca, A., Lukichev, S. and Wagner, G. (2006) 'Modeling web services with URML', in *Proceedings of SBPM2006*, 11 June, Budva, Montenegro.

Goedertier, S. and Vanthienen, J. (2006) 'Designing compliant business processes with obligations and permissions', *Business Process Management Workshops Lecture Notes in Computer Science*, Vol. 4103/2006.

Knolmayer, G., Endl, R. and Pfahrer, M. (2000) 'Modeling processes and workflows by business rules', in Aalst, W.M., Desel, J. and Oberweis, A. (Eds.): *Business Process Management, Models, Techniques, and Empirical Studies, Lecture Notes in Computer Science*, Vol. 1806, pp.16–29, Springer-Verlag, London.

Koshkina, M. and Van Breugel, F. (2004) 'Modeling and verifying web service orchestration by means of the concurrency workbench', *ACM SIGSOFT Software Engineering Notes*, September, Vol. 29, No. 5.

Lee, S., Kim, T.Y., Kang, D., Kim, K. and Lee, J.Y. (2003) 'Composition of executable business process models by combining business rules and process flows', in *Expert Systems with Applications*, July 2007, Vol. 33, No. 1, pp.221–229.

Li, X. and Marín, J.M. (2004) 'Composite event specification in active database systems: a Petri nets approach', in Jensen, K. (Ed.): *Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, 8–11 October, Aarhus, Denmark.

Lu, R. and Sadiq, S. (2007) *A Survey of Comparative Business Process Modeling Approaches*, *Business Information System Lecture Notes in Computer Science*, Vol. 4439/2007.

Müller, R., Greiner, U. and Rahm, E. (2004) 'AgentWork: a workflow system supporting rule-based', *Workflow Adaptation. In Data & Knowledge Engineering*, Vol. 51, No. 2, pp.223–256

Object Management Group (OMG) (2008) 'Semantics of business vocabulary and business rules (SBVR)', available at http://www.omg.org/spec/SBVR/1.0/PDF.

Object Management Group (OMG) (2009) 'Business process model and notation (BPMN)', in OMG Document Number: dtc/2009-08-14.

Ouyang, C., Verbeek, E., Van der Aalst, W.M.P., Breutel, S., Dumas, M. and ter Hofstede, A.H.M. (2005) 'WofBPEL: a tool for automated analysis of BPEL processes', in *ICSOC 2005, International Conference of Service-Oriented Computing*, Berlin.

Pesic, M., Schonenberg, H. and Van der Aalst, W.M.P. (2007), 'DECLARE: full support for loosely-structured processes', in the *Eleventh IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*.

Pu, G., Zhao, X., Wang, S. and Qiu, Z. (2005) 'Towards the semantics and verification of BPEL4WS', in *Theoretical Computer Science*, July, pp.33–52, Elsevier, New Castle, UK.

Regev, G., Soffer, P. and Schmidt, R. (2006) 'Taxonomy of flexibility in business processes', *Seventh Workshop on Business Process Modeling, Development, and Support in Conjunction with CAiSE'06*.

Russell, N., Van der Aalst, W.M.P. and ter Hofstede, A.H. (2006) *Exception Handling Patterns in Process-Aware Information Systems*, BPM Center Report BPM-06-04, BPMcenter.org.

Schroeder, M. and Wagner, G. (2002) 'Languages for business rules on the semantic web', in *Proc. of the Int. Workshop on Rule Markup*, June, Vol. 60, CEUR-WS Publication, Italy.

Taveter, K. and Wagner, G. (2001) 'Agent-oriented business rules: Deontic assignments', in *Proc. of Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations (OES-SEO2001)*.

Van Breugel, F. and Koshkina, M. (2006) 'Models and verification of BPEL', Research report, Department of Computer Science and Engineering, University of York, September, available at http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf.

Van der Aalst, W.M. (1998) 'The application of Petri nets to workflow management', *The Journal of Circuits, Systems and Computers*, Vol. 8, No. 1, pp.21–66.

Van Eijndhoven, T., Iacob, M.E. and Ponisio, M.L. (2008) *Achieving Business Process Flexibility with Business Rules*, EDOC 2008: 95-104.

Wagner, G. (2005) 'Rule modeling and markup', in Eisinger, N. and Maluszynski, J. (Eds.): *Reasoning Web*, Vol. 3564, pp.251–274, Springer, Msida, Malta.

Wagner, G., Giurca, A. and Lukichev, S. (2006) 'Modeling web services with URML', in *Proceedings of Semantics for Business Process Management Workshop*, June, Budva, Montenegro.

Xiao, H., Guo, J. and Zou, Y. (2007) 'Supporting change impact analysis for service oriented business applications', *Proceedings of the International Workshop on Systems Development in SOA Environments (SDSOA'07)*, IEEE Computer Society, Washington, DC, USA.

Yang, Y., Tan, Q., Yu, J. and Liu, F. (2005) 'Transformation BPEL to CP-nets for verifying web services composition', in the *International Conference on Next Generation Web Services Practices*, Korea.

Zur Muehlen, M., Indulska, M. and Kamp G. (2007) 'Business process and business rule modeling: a representational analysis', *3rd International Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE 2007)*, Annapolis, Maryland, USA.

Notes

1 Available at http://www.workflowpatterns.com.

Appendix 1

RbBPDL expressiveness

To study the expressiveness of the RbPDL language, we check, in the following table, whether it is possible to realise a subset of the workflow pattern[1] using this language. If the language supports directly the pattern through one of its constructs, it is rated +. If the pattern is not directly supported, it is rated.

Table A1 Support for the workflow patterns in RbBPDL

Control-flow patterns	*Support*	*Explication*
Sequence	+	Sequence pattern is supported by defining a rule's post-event which activates another rule
Parallel split	+	Parallel split pattern is supported by defining a rule's post-event with active multiple rules simultaneously
Synchronisation	+	Synchronisation pattern is supported by defining the a rule activation event as conjunction of a set of post-event rules
Exclusive choice	+	Exclusive choice pattern is supported by defining a rule's post-event that activates two rules that have two contradictory conditions
Cancel activity	+	Cancel activity pattern is supported by using the with the CANCEL instruction with set 'cancel all activities' option to false
Cancel case	+	Cancel case pattern is supported by using the with the CANCEL instruction with set 'cancel all activities' option to true
Implicit termination	+	Implicit termination pattern is supported since the process is terminated when there are no rules to trigger
Explicit termination	+	Explicit termination pattern is supported since the process is terminated when an event that belongs to the end events categories is triggered
Data patterns	*Support*	*Explication*
Workflow data	+	Workflow data pattern is supported because all process variables are global variables
Task data	-	Not supported
Task to task	+	Task to task pattern is supported because the data transfer between one activity to another is possible by using the same parameters of input/output or using the COPY instruction
Task to environment	+	Task to environment pattern is supported because the data transfer between one activity to an external environment is possible by using the COPY instruction
Data transfer by value	-	Not supported
Data transfer by reference	+	Data transfer by reference pattern is supported because variables defined in the RbBDPL instruction are the references to variables declared in <Variable> part
Task precondition	+	Task precondition pattern is supported using the condition of the rule
Task post-condition	+	Task precondition pattern is supported using the post-condition of rule
Resource patterns	*Support*	*Explication*
Direct allocation	+	Direct allocation pattern is supported by involving participants in a business activity
Deferred allocation	+	Deferred allocation pattern is supported using DISCOVER instruction.
Role-based allocation	+	Role-based allocation pattern is supported by defining the role of each participant
Delegation	-	Not supported

Appendix 2

BPFAMA prototype

Figure 14 BPFAMA rules definer interface (see online version for colours)

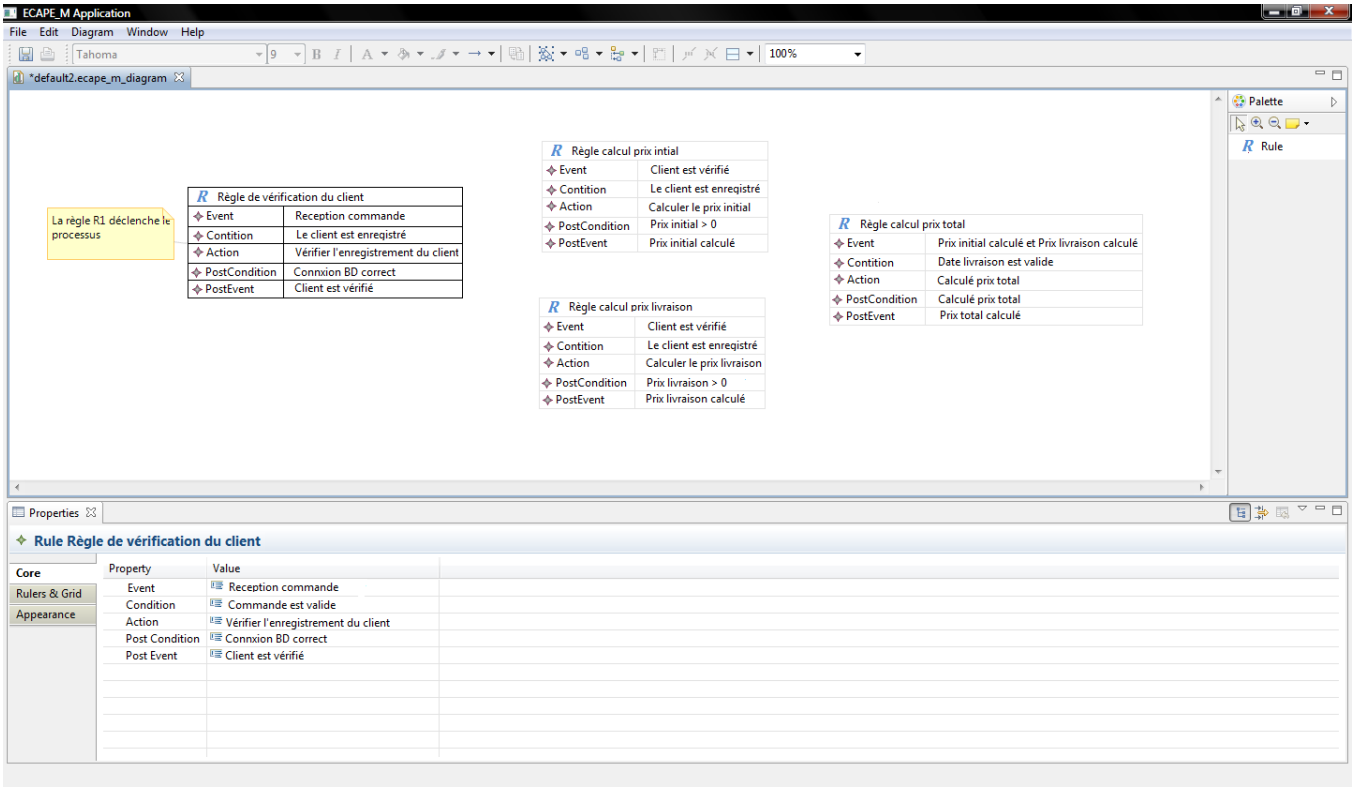


Figure 15 BPFAMA rules behaviour analyser interface (see online version for colours)

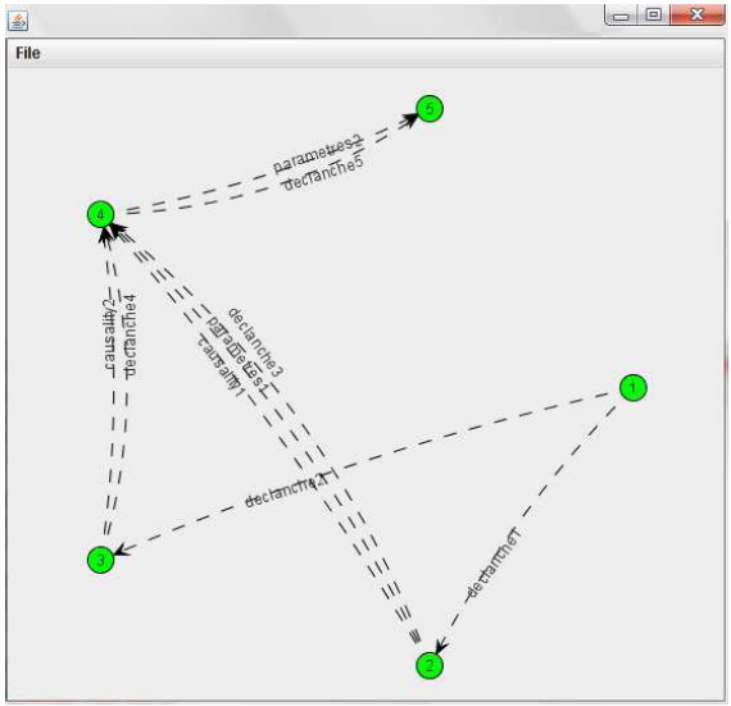


Figure 16 ECAPE net conversion algorithm