A Data-driven Workflow Based on Structured Tokens Petri Net

Nahla HADDAR
MIRACL/FSEGS laboratory :Computer Department
FSEGS, Airport avenue Km 4, BP 1013 Sfax 3018, Tunisia
Email :nhaddar@ymail.com

Mohamed TMAR, Faiez Gargouri
MIRACL/ISIMS laboratory :Computer Department
ISIMS, B.P. Technology Center:
242 Sfax 3021, Tunisia
Emails :mohamed.tmar@isimsf.rnu.tn, faiez.gargouri@gmail.com

***Abstract*—Business processes design and implementation within a company are mainly based on the specification of actors and their different tasks. Data and general information are transmitted in a very specific organization among actors, applications and the information system, which constitute a workflow. In this paper, we present an approach for workflow process modeling. The model is in charge of representing both control flow and shared data in the workflow process, and it can be analysed to verify its correctness before implementation. This workflow modeling approach has been implemented into $Opus$ system that provides a set of graphical interfaces to model and execute the business process tasks. The system also provides a workflow engine that grants automatic workflow processing by interpreting the workflow process.**

***Keywords*—Workflow modeling; Workflow management system; Petri Nets; Data-driven approach; Structured token.**

I. Introduction

At the beginning of this century, workflow management concentrated on the design and documentation of business process [1]. Therefore, it focused on the dependencies between tasks and their sequencing, while data and resources played a very minor role. Many new approaches have been introduced, e.g, Petri Nets [2], Business Process Modeling Notation (BPMN) [3], Business Process Execution Language (BPEL) [4], etc.; but only a few of them are of ongoing interest in modeling the exchanged data flow in the business process. Moreover, the importance of data in business processes has increased progressively in recent years with the appearance of the data-driven approaches.

As execution and expressiveness have got more attention, also validation of the workflow model has needed to get attention. One big standard in this attribute is Petri Nets. Petri Nets are currently among the best known techniques for workflows specification [5].

In this paper, we present a formal approach inspired from the data-driven approach and the Petri Net formalism to model workflow processes. The resulting model can be analyzed for validation and automatically generated by the workflow engine for process execution.

The rest of the paper is organized as follows :we illustrate the related work in Section II and then, we elucidate our approach for workflow modeling in Section III. We illustrate in Section IV the possible information flows routing. Then, we demonstrate our approach by an example of workflow model in Section V. In Section VI, we explain how our workflow model can be analyzed and verified and we present our workflow management system $Opus$ in Section VII. Section VIII concludes the paper.

II. Related Work

Many new approaches have emerged, which shifted their focus to combination of data flow and control flow. An emerging approach uses artifacts, that combine data and process by using atrifacts and Petri Nets model, is the Business Artifacts (BA) [6].

The BA approach focuses on solving decision problems, related to reachability, avoiding dead-ends and redundancy, but it does not provide a graphical notation for process modeling. Despite it was formally defined, the BA does not provide a formal mechanism for process verification. Process verification has been widely studied in workflow research, with states machines in Petri Nets [7], [8], graphs [9], data dependencies [10], etc.

Another formal approach based on Petri Nets model is the CorePro Framework [1]. The CorePro enables to model the data-driven specification and then, to create automatically the process structures based on given data structures in the model level. As well, CorePro provides some simple rules to verify the soundness properties of the data-driven process structures. However, it has skipped to retain the object states which have already been activated before the execution.

Many extensions of Petri nets in which tokens carry data have been defined in the literature, in order to improve expressiveness of workflow models. Data Nets (DN) [11] are an extension of Petri nets in which tokens are taken from a linearly ordered and dense domain, and transitions can perform whole place operations like transfers, resets or broadcasts. Although, a data net can be viewed as a constrained multiset rewriting systems (CMRS) enriched with whole-place operations. And, according to researches developped in [12], whole-place operations augment the expressive power of Petri nets only in the case of black indistinguishable tokens, but not for models in which tokens carry data taken from an ordered domain. Weakness refers here to the fact that the CMRS encoding simulates a lossy version of data nets, e.g., data nets in which tokens may get lost.

All the approaches mentioned above focus on the data

 ISBN: 978-1-61208-230-1

routing and data managed by the process, but they consider activities as black-boxes in which application data is managed by invoked application components. Some of them, like DN, can apply transitions that read from or write to some data element, but with limited power to manage all the handled data element. This is why processes have to be modeled at a higher-level of abstraction to reflect the preferred work practice.

III. Workflow Modeling Using Structural Petri Net Tokens

We have inspired from Petri Nets to propose a new workflow modeling approach leading to a workflow process model. To manage all the data handled by the work procedures, we use the notion of data-driven process structures.

So, we describe the process by respective data structures, and we define a data structure as a pair $s = (C,\ D)$, where C is a list of attributes and D is a list of tuples, each tuple is an ordered set of attribute values. Formally, $\forall n,\ m \in \mathbb{N}$:

$$C = (c_1, c_2 \dots c_n)$$
$$D = \{(d_{1_1}, d_{1_2} \dots d_{1_n}), (d_{2_1}, d_{2_2} \dots d_{2_n}) \dots (d_{m_1}, d_{m_2} \dots d_{m_n})\}$$

Each attribute c_i is an ordered pair of attribute name n_i and type name t_i, such as :

$\forall i, t_i \in \{SmallInt,\ Int,\ BigInt,\ Float,\ Double,\ Real,\ Decimal,\ Char,\ Varchar,\ Text,\ Date,\ Year,\ Boolean\}$

$\forall i, j,\ d_{i_j} \equiv t_j$:an attribute value is a specific valid value for the type of the attribute.

The workflow process is defined as a Petri Net representing the work, where a place corresponds to a data structure that contains structured tokens (tuples) and a transition corresponds to a task. A workflow is then a quadruplet $WF = (S,\ T,\ Pre,\ Post)$ where :

- S is a finite set of data structures,
- T is a finite set of tasks,
- $Pre\ : S \times T \to \mathbb{N}$ is the pre-incidence matrix,
- $Post\ : T \times S \to \mathbb{N}$ is the post-incidence matrix.

A workflow process is defined by an oriented net with two node types representing $data\ structures$ and $tasks$ manipulating the tuples of these structures. A task consumes data structure tuples to produce others, which can then be consumed by other tasks.

A task t is said to be enabled if each input data structure $s \in S$ is marked with at least x_i tuples (refers to $Pre(s,\ t)$, which defines the weight of the edge from s to t). A firing of an enabled task t consumes x_i tuples from each input data structure s, and produces x_j tuples (refers to $Post(t,\ s)$) to each output data structure of t. $Post(t,\ s)$ is the weight of the edge from t to s.

We have to clarify that in our case we cannot be limited to a simple post-incidence matrix. In fact, each transition consumes an undefined number of tuples and produces a number belonging to a well determined range, depending on its processing (See Table I, in Appendix). For example, if a transition is a tuples union operation of two data structure s_1 and s_2 containing respectively x_1 and x_2 number of tuples, it will produce a number of tuples belonging to the interval :max($x_1,\ x_2$) and $x_1 + x_2$ (because the union operation eliminates the duplicated tuples).

So, we define two post-incidence matrices :$Post_{Min}$ and $Post_{Max}$, as a values interval which limits all possible post-incidence matrices. Formally :

$\forall t \in T$ and $s \in S,\ Post_{Min}(t,\ s)$:is the edge going from transition t to place s minimal weight.

$\forall t \in T$ and $s \in S,\ Post_{Max}(t,\ s)$:is the edge going from transition t to place s maximal weight.

$\forall t \in T$ and $s \in S,\ Post(t,\ s) \in [Post_{Min}(t,\ s),\ Post_{Max}(t,\ s)]$.

We explain this idea in details through the example in Figure 1.

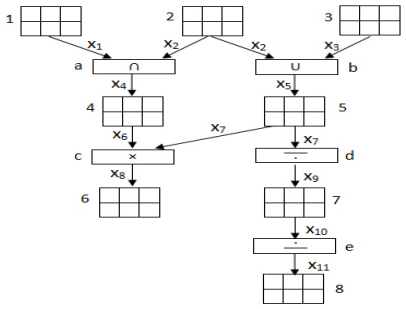

Figure 1. Example of workflow model

The example illustrated by Figure 1 contains eight places ($s_1,\ s_2 \dots s_8$) and five transitions ($t_a,\ t_b \dots t_e$). Each edge is associated with a weight ($x_i > 0$).

We define its Pre matrix by :

$$Pre = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccc} t_a & t_b & t_c & t_d & t_e \end{array} \\ \begin{pmatrix} x_1 & 0 & 0 & 0 & 0 \\ x_2 & x_2 & 0 & 0 & 0 \\ 0 & x_3 & 0 & 0 & 0 \\ 0 & 0 & x_6 & 0 & 0 \\ 0 & 0 & x_7 & x_7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_{10} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Following the example illustrated in Figure 1, and the definition of its transitions in Table I, we can determine the values range of output tokens for each fired transition as follows :

- $x_4 \in [0,\ min(x_1, x_2)]$
- $x_5 \in [max(x_2, x_3), x_2 + x_3]$
- $x_8 = x_6 \times x_7 \in [x_6 \times x_7,\ x_6 \times x_7]$
- $x_9 = x_7 \in [x_7,\ x_7]$
- $x_{11} = x_{10} \in [x_{10},\ x_{10}]$

So, we can deduce the matrices :

$Post_{Min}$:

$$\begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccc} t_a & t_b & t_c & t_d & t_e \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & max(x_2, x_3) & 0 & 0 & 0 \\ 0 & 0 & x_6 \times x_7 & 0 & 0 \\ 0 & 0 & 0 & x_7 & 0 \\ 0 & 0 & 0 & 0 & x_{10} \end{pmatrix} \end{array}$$

$Post_{Max}$:

$$\begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccc} t_a & t_b & t_c & t_d & t_e \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ min(x_1, x_2) & 0 & 0 & 0 & 0 \\ 0 & x_2 + x_3 & 0 & 0 & 0 \\ 0 & 0 & x_6 \times x_7 & 0 & 0 \\ 0 & 0 & 0 & x_7 & 0 \\ 0 & 0 & 0 & 0 & x_{10} \end{pmatrix} \end{array}$$

Using these three matrices (Pre, $Post_{Min}$ and $Post_{Max}$) we can derive several properties of the designed workflow model to be verified. We detail this idea in Section VI.

To reach the lowest level of abstraction, we need algebra over data structures. So, we have inspired from the relational algebra to define the tasks needed to produce data structures from others. As illustrated in Table I, we redefine the relational algebra operations in a formal way in order to suit the Petri Net formality. To keep equivalence between the attributes of data structures assigned to operations as Union, Difference, Intersection, and Division, we define the Permutation and Substitution operations.

Furthermore,we suggest an Extension operation to add attributes in a structure scheme, where its values are generated through applying a function. And finally, to insert data structure tuples in a data structure, we define the Alimentation operation.

As for example, we explain the Projection operation illustrated in Table I by the following example :

Whether the structure $Products = (C_j,\ D_j)$, where :

$C_j = (Id,\ designation,\ price,\ Stock)$,

$D_j = \{(1,\ aa,\ 20.5,\ 1000),\ (2,\ ab,\ 25.0,\ 2500),\ (3,\ ac,\ 22.75,\ 1500)\}$.

$Prod = (C_i,\ D_i) = \div(Products,\ b)$ such as $b = (1,\ 0,\ 0,\ 1)$.

So, q is defined as following :

$$q = \sum_{k=1}^{5} b_k = 2 \quad \Rightarrow C_i = (c_{j_{j'_1}},\ c_{j_{j'_2}})$$

$$j'_1 = \min_{l = \{1,2\ldots5\}} \left(l = 1 \quad \Rightarrow c_{i_1} = c_{j_1} = Id \right)$$
$$\sum_{p=1}^{l} b_p = 1$$

$$j'_2 = \min_{l = \{1,2\ldots5\}} \left(l = 4 \quad \Rightarrow c_{i_2} = c_{j_4} = Stock \right)$$
$$\sum_{p=1}^{l} b_p = 2$$

$$\Rightarrow Prod = ((Id,\ Stock), \{(1,\ 1000), (2,\ 2500), (3,\ 1500)\})$$

IV. Information Flows Routing

Our workflow model can express sequential, conditional and parallel routing flow.

$Sequential$ routing is used to deal with causal relationships between tasks [8]. Figure 2 shows that sequential routing can be modeled by our operations graph.

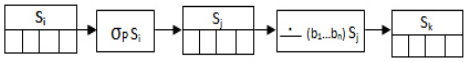

Figure 2. Sequential routing

$Parallel$ routing is used where two tasks B and C have to be executed at the same time. To model parallel routing, two building blocks are identified :The AND-Split and the AND-Join [8]. Figure 3 shows that both building blocks can be modeled by our operations graph.

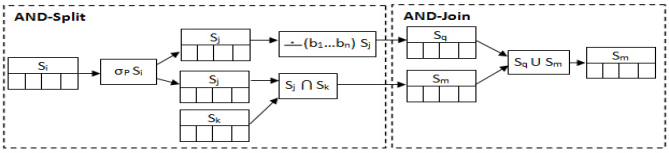

Figure 3. Parallel routing

$Conditional$ routing is used when there is a mutual execution between two tasks according to a condition. We can express conditional routing by a simple network using $control$ operations.

Indeed, the control operation decides to continue, or not, the information flows routing according to the controlled data structure content. Whether s_i is the controlled data structure, s_j is the data structure expected by the next transition if the condition is verified, so, s_i will be controlled by one of the $control$ operations which are defined as follows :

Control operation 1, noted $\pm$:

$$s_i \ \pm s_j = \begin{cases} s_i \ if \ s_j = \phi \\ \phi \ otherwise \end{cases}$$

Control operation 2, noted $\mp$:

$$s_i \ \mp s_j = \begin{cases} s_i \ if \ s_j \neq \phi \\ \phi \ otherwise \end{cases}$$

An example of control flow is illustrated in Figure 5 in Appendix, where the structure s_6, which contains all the unpaid bills of the current customer, is used by task t_5 to decide the customer solvency. So, if s_6 contains one or more tokens, t_5 will decide that the customer is not solvent, and it will finish the order management process. Otherwise, t_5 will reproduce s_2 tokens in s_7 in order to be sent to Inventory Check Role.

V. Example Modeled Using our Approach

Consider an office procedure for order processing within a company. When a customer sends his order by email, the job is sent to the customer solvency check, and then to the inventory check. After the evaluation, either a rejection mail is sent to the customer, or the order is sent to shipping and billing. In this paper, we restrict our example to the solvency check and the inventory check processes.

To simplify the representation of the model, we group the tasks related to the same function in the company according to roles. So, each role work is presented by a sub-process belonging to the whole workflow process definition.

As shown in Figure 5, when a customer mail arrives, the

 ISBN: 978-1-61208-230-1

workflow will launch. S_1 tokens (present customers data) are to be consumed by t_1 in order to select the current customer (CC) information by his first name and his last name (the selection condition is to be seized by the Solvency Check Role (SCRole) during the execution of the workflow).

The resulted structure s_2 token (s_2 contains only one token presenting the CC information), and the s_3 tokens (present bills data of customers) are to be used by t_2 to produce a single data structure containing bills data, and the CC information. Resulted structure s_4 tokens are to be used by t_3 to create an inner join between bills data and the CC, in order to select only the CC bills. So, s_5 tokens present the CC history on bill payment. To check customer solvency, t_4 selects only s_5 tokens which have a paid attribute value equals to false. The resulted structure s_6 is to be then used to decide the customer solvency. Task t_5 is a control operation, which verifies s_6 content. If s_6 contains one or more tokens, t_5 will decide that the customer is not solvent (because he has unpaid bills), and it will finish the order management process. Otherwise, t_5 will reproduce s_2 tokens in s_7 in order to be sent to Inventory Check Role (ICRole).

To select the ordered products, t_6 extends s_8 (contains all products data) by the *ord_qtity* attribute (accepts only integer values), in order to allow the ICRole to seize the ordered quantities relatively to the ordered products. Then, t_7 selects from the resulted structure s_{10} only tokens having an ordered quantity value higher than zero and lower than the stocked product quantity. The resulted tokens are stocked in s_{11}.

In parallel, t_8 applies a projection operation on s_7, to get the structure s_9 having as a token, the CC identifier. If there are available ordered products, the ICRole has to create a new order. To verify availability, we define the control operation in t_9. If s_{11} contains one or more tokens (there is, at least, one available product), t_9 will reproduce s_9 token in s_{12}, then, t_{10} will add a new order in s_{13}. It remains to create the new order lines. So, the ICRole has to seize the new order identifier, t_{13} will save his seizure in s_{17}. Then, t_{14} will create the new order lines by applying a simple inner product between s_{17} token and s_{15} tokens (present identifiers of the ordered products and their relative ordered quantities).

VI. The Workflow Verification

We provide techniques based on Pre and Post matrices, to ensure that WF satisfies the minimum requirements for correctness.

First of all, we verify that each data structure is the result of at most a single transition. Formally, consider n places and m transitions in the workflow model :$\forall\ i \in \{1,\ 2 \dots n\}$,

$$\forall\ i \in \{1,\ 2 \dots n\},\ |j \in \{1,\ 2 \dots m\},\ Pre(s_i,\ t_j) \neq 0| \leq 1. \quad (1)$$

To explain Equation 1, we resume the example in Figure 1, and we verify s_4. So, for $i = 4$:

$|j \in \{a \dots e\},\ Pre(s_4,\ t_j) \neq 0| = |x_6| = 1 \ \Rightarrow\ s_4$ verifies the condition.

In the rest of this section we focus on the verification of liveness property of the model. For us, to verify this property, we have to begin with defining the initial and the final marking of WF.

Formally, the initial marking i is defined as : $i = \begin{pmatrix} i_1 \\ i_2 \\ \dots \\ i_n \end{pmatrix}$, such as :$\forall j \in \{1,\ 2, \dots n\}$

$$i_j = \begin{cases} \max\limits_{k \in \{1,\ 2, \dots m\}} Pre(s_j,\ t_k), \\ \quad if\ \forall l \in \{1,\ 2, \dots m\} Post_{Max}(s_j,\ t_l) = 0. \\ 0, \text{ otherwise.} \end{cases} \quad (2)$$

We explain Equation 2 using the example in Figure 1 :

$\forall j \in \{1 \dots 8\}$, the condition $\forall l \in \{a \dots e\} Post_{Max}(s_j,\ t_l) = 0$ return $true$ only for $j = 1$, $j = 2$ and $j = 3$. So, $\forall j \in \{4 \dots 8\}$, $i_j = 0$.

For $j = 1$: $\max\limits_{k \in \{a \dots e\}} Pre(s_1,\ t_k)$ return $Pre(s_1,\ t_a) = x_1$.

$\Rightarrow\ i_1 = x_1$

For $j = 2$: $\max\limits_{k \in \{a \dots e\}} Pre(s_2,\ t_k)$ return $Pre(s_2,\ t_a) = x_2$ (or $Pre(s_2,\ t_b) = x_2$).

$\Rightarrow\ i_2 = x_2$

For $j = 3$: $\max\limits_{k \in \{a \dots e\}} Pre(s_3,\ t_k)$ return $Pre(s_3,\ t_b) = x_3$.

$\Rightarrow\ i_3 = x_3$

As we define an interval for Post matrices, we define an interval for final possible markings. Formally, $\forall j \in \{1,\ 2, \dots n\}$: A minimal final marking o^- is defined as :$o^- = \begin{pmatrix} o_1^- \\ o_2^- \\ \dots \\ o_n^- \end{pmatrix}$, where :

$$o_j^- = \begin{cases} \max\limits_{k \in \{1,\ 2, \dots m\}} Post_{Min}(s_j,\ t_k), \\ \quad if\ \forall l \in \{1,\ 2, \dots m\} Pre(s_j,\ t_l) = 0. \\ 0,\ otherwise. \end{cases} \quad (3)$$

Let us calculate o^- of the model in Figure 1 :$\forall j \in \{1 \dots 8\}$, the condition $\forall l \in \{a \dots e\} Pre(j,l) = 0$ return $true$ only for $j = 6$, $j = 8$. So, $\forall j \in \{1 \dots 8\} \backslash \{6,8\}$, $o_j^- = 0$.

For $j = 6$:

$\max\limits_{k \in \{a \dots e\}} Post_{Min}(s_6,\ t_k)$ return $Post_{Min}(s_6,\ t_c) = x_6 \times x_7$.

$\Rightarrow\ o_6^- = x_6 \times x_7$

For $j = 8$:

$\max\limits_{k \in \{a \dots e\}} Post_{Min}(s_8,\ t_k)$ return $Post_{Min}(s_8,\ t_e) = x_{10}$.

$\Rightarrow\ o_8^- = x_{10}$

The maximal final marking o^+ is defined as o^- but with using the $Post_{Max}$ matrix instead of the $Post_{Min}$:

$o^+ = \begin{pmatrix} o_1^+ \\ o_2^+ \\ \dots \\ o_n^+ \end{pmatrix}$, such as :

$$o_j^+ = \begin{cases} \max\limits_{k \in \{1,\ 2, \dots m\}} Post_{Max}(s_j,\ t_k), \\ \quad if \forall l \in \{1,\ 2, \dots m\} Pre(s_j,\ t_l) = 0. \\ 0,\ otherwise. \end{cases} \quad (4)$$

Assuming i as the initial state, o as the final state of a process, the workflow model is live if and only if :

- For every state M reachable from state i, there exists a firing sequence leading from state M to state o [8]. We adopt this rule to WF by applying the following rule : Whether R^+ (resp. R^-) is the net presenting the maximal (resp. minimal) output function, such as :
$$R^+ = (S,\ T,\ Pre,\ Post_{Max}),$$
$$R^- = (S,\ T,\ Pre,\ Post_{Min}),$$
$$\forall_M (i \overset{*}{\rightarrow} R^+(M)) \Rightarrow (R^+(M) \overset{*}{\rightarrow} o^+),$$
$$(i \overset{*}{\rightarrow} R^-(M)) \Rightarrow (R^-(M) \overset{*}{\rightarrow} o^-). \quad (5)$$
- There are no dead transition in the workflow model [8]. We also adopt this rule to WF by applying the following rule :
$$\forall_{t \epsilon T} \exists_{M,M'}, (i \overset{*}{\rightarrow} R^+(M) \overset{t}{\rightarrow} M'). \quad (6)$$

We define the simple algorithm below to ensure the verification of Equations 5.

Algorithm 1 Verification

/* t is a task to verify */
Procedure VerificationT(t)
VerificationT(t)= $\bigwedge_{\substack{i \in \{1,\ 2 \ldots n\} \\ Pre(s_i,t) \neq 0}}$ VerificationS(s_i);
Procedure VerificationS(s)
/* s is a root node */
if $\forall j \in \{1,\ 2 \ldots m\},\ Post_{Max}(s,\ t_j) = 0$ **then**
VerificationS(s)=true;
else
/* t_i is the task which has $Post_{Max}(s,\ t_i) \neq 0$ */
VerificationS(s)=VerificationT(t_i) where $Post_{Max}(s,\ t_i) \neq 0$;
end if

We apply Algorithm 1 on the example illustrated in Figure 1, and we choose to verify task t_e since its output data structure is a final state in the model; so, its verification generates the verification of all firing sequences leading from a state M to this final state.

VerificationT(t_e)= $\bigwedge_{\substack{i \in \{1,\ 2 \ldots 8\} \\ Pre(s_i,t_e) \neq 0}}$ VerificationS(s_i)
= VerificationS(s_7) = VerificationT(t_d)
= VerificationS(s_5) = VerificationT(t_b)
= VerificationS(s_2) $\wedge$ VerificationS(s_3)
= true $\wedge$ true = true.

To verify Equation 6, we have to verify that the model is without structural conflicts. we assume that WF has a structural conflict if it contains at least two tasks t_i and t_j having the same input data structure s. As the case in Figure 1, the model has a structural conflict caused by t_b and t_c which share s_2. To avoid these cases, we extend the model by adding extra tasks $T^* = \{t_{copy_1}, t_{copy_2} \ldots t_{copy_k}\}$ such as k is the number of data structures which cause conflicts, and t_{copy} is a $Copy$ operation (See Table I), which allow to create copies from a shared data structure to satisfy the need of tasks in a conflict.

The extended model $WF_+ = (S_+,\ T_+,\ Pre_+,\ Post_+)$ is defined as follows :$S_+ = S$, $T_+ = T \cup T^*$, $Pre_+ = S \times T_+$ and $Post_+ = T_+ \times S$.

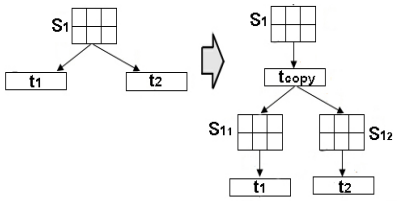

Figure 4. Removing the conflict

So, to resume, Algorithm 1 can verify that WF_+ is live.

VII. Implementation of the Workflow Management System $Opus$

The $Opus$ workflow system consists of a number of components including a workflow engine and a Petri Net editor. Workflow specifications can be designed using the $Opus$ editor and deployed in the $Opus$ engine for execution.

The $Opus$ engine follows the workflow model definition and interprets automatically the code executing the workflow. Then it invites each role to perform its tasks according to its feasibility and urgency. The verification of the conceived model is automatically ensured as follows in Algorithm 1 and Equation 1. To integrate workflows with the Information System (IS), we developed some tools, e.g., the Import tool (it imports a table tuples to a definite data structure belonging to the workflow process), the ImportId tool (it imports the tuple identifier of the last tuple inserted in a definite table), the Insert tool (it inserts data structure tuples in a definite IS table) and the Update tool (it updates a table in the IS with a data structure tuples). To perform these operations, and operations which requires two identical data structure schemes, $Opus$ system is equipped with a matching tool, which uses the Substitution and the Permutation operations.

VIII. Conclusion and Future Work

The proposed approach is modular in a sense that the workflow process is to be decomposed on sub-processes which facilitates any eventual updates on the workflow process model. In fact, the changes related to the evolution in the role work, causes the change of its sub-process without damaging other sub-processes. In particular, the detailed formal definition of tasks and data structures is useful for the *Opus* engine, to extract all the process specifications. However, this approach must be completed by many functionalities. In fact, we plan to provide techniques to verify others Petri Nets property, like boundness, soundness, etc. We also plan to implement a simulation tool to decision-makers, in order to improve the business process, and a module for documents generation (invoice, purchase order, etc.) :the system can manage the content but not the container.

 ISBN: 978-1-61208-230-1

REFERENCES

[1] D. Müller, M. Reichert, and J. Herbst, "Data-driven modeling and coordination of large process structures," in *OTM Conferences (1)*, 2007, pp. 131–149.

[2] C. Petri, "Communications with automata," Ph.D. dissertation, Institut für instrumentelle Mathematik, Bonn, 1962.

[3] Object Management Group (OMG), "Business process model and notation (bpmn)(version 2.0)," Tech. Rep., 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/

[4] OASIS WSBPEL Technical Committee, "Web services business process execution language version 2.0," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[5] W.V.D. Aalst, J.M. Colom, F. Kordon, G. Kotsis, and D. Moldt, *Petri Net Approaches for Modelling and Validation*, ser. Lincom Studies in Computer Science, 2003, vol. 1.

[6] A. Nigam and N.S. Caswell, "Business artifacts: An approach to operational specification," *IBM Syst. J.*, vol. 42, no. 3, pp. 428–445, July 2003.

[7] W.V.D Aalst, "Verification of workflow nets," in *Application and Theory of Petri Nets 1997*, P. Azéma and G. Balbo, Eds., vol. 1248, 1997, pp. 407–426.

[8] W.V.D Aalst, "The application of petri nets to workflow management," *Journal of Circuits, Systems, and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[9] W. Sadiq and M.E. Orlowska, "Analyzing process models using graph reduction techniques," *Information Systems*, vol. 25, pp. 117–134, 2000.

[10] S.X. Sun, J.L. Zhao, J.F. Nunamaker, and O.R.L. Sheng, "Formulating the data-flow perspective for business process management," *Information Systems Research*, vol. 17, no. 4, pp. 374–391, 2006.

[11] R. Lazic, T.C. Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell, "Nets with tokens which carry data," *Fund. Informaticae*, vol. 88, no. 3, pp. 251–274, 2008.

[12] P.A. Abdulla, G. Delzanno, and L. Van Begin, "A language-based comparison of extensions of petri nets with and without whole-place operations," in *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications*, ser. LATA '09. Springer-Verlag, 2009, pp. 71–82.

APPENDIX

Table I
OPERATIONS DEFINITION

Operation	Formal definition
Named :Inner Product Description :Performs the combination of all structure tuples with those of another structure. Noted:$\times$	$\forall\ s_j = (C_j,\ D_j),\ s_k = (C_k,\ D_k)$ $C_j = (c_{j_1},\ c_{j_2} \dots c_{j_{n_j}})$, $C_k = (c_{k_1},\ c_{k_2} \dots c_{k_{n_k}})$ $s_i = s_j \times s_k$ $\Rightarrow s_i = ((c_{j_1} \dots c_{j_{n_j}}, c_{k_1} \dots c_{k_{n_k}}), D_i)$ Where : $D_i = \bigcup_{\substack{l \in \{1 \dots n_j\} \\ p \in \{1 \dots n_k\}}} \{(d_{j_l{}^1} \dots d_{j_l{}^{n_j}}, d_{k_p{}^1} \dots d_{k_p{}^{n_k}})\}$ Resulted tokens number :$x_i = x_j \times x_k$
Named :Selection Description :Selects only the structure tuples that meet the desired criteria. Noted :σ	Whether P is the selection property, $\forall\ s_j = (C_j,\ D_j), s_i = \sigma_P s_j$ $\Leftrightarrow s_i = (C_j,\ \bigcup_{\substack{e \in D_j \\ P(e)}} \{e\})$ Resulted tokens number :$x_i \in [0,\ x_j]$
Named :Difference Description :Subtracts the tuples of a data structure from another one. Noted :$-$	$\forall\ s_j = (C,\ D_j),\ s_k = (C,\ D_k)$ $\Rightarrow\ s_j - s_k = (C,\ D_j - D_k)$ Resulted tokens number : $x_i \in [x_j - x_k,\ x_j]$
Named :Projection Description :Selects only the structure columns (attributes) that we are interested in. Noted :$\div$	$s_j = (C_j, D_j), \forall (b_1 \dots b_n) \in \{0,1\}^n$ $s_i = (C_i, D_i) = \div_{(b_1 \dots b_n)} s_j$ Where c_i is a selected (resp. not selected) attribute, if $b_i = 1$ (resp $b_i = 0$). $\Leftrightarrow \begin{cases} C_i = (c_{j_{j'_1}},\ c_{j_{j'_2}} \dots c_{j_{j'_q}}) \\ D_i = \{(d_{j_{1_{j'_1}}},\ d_{j_{1_{j'_2}}} \dots d_{j_{1_{j'_q}}}), \\ (d_{j_{2_{j'_1}}},\ d_{j_{2_{j'_2}}} \dots d_{j_{2_{j'_q}}}) \dots (d_{j_{m_{j_{j'_1}}}}, \\ d_{j_{m_{j_{j'_2}}}} \dots d_{j_{m_{j_{j'_q}}}})\} \end{cases}$ Such as : $q = \sum_{k=1}^{n} b_k$:is the number of attributes in the structure result. And : $j'_k = \min_{\substack{l = \{1,2 \dots n\} \\ \sum_{p=1}^{l} b_p = k}} l$:refers to the projection attributes indices. Resulted tokens number : $\begin{cases} x_i = 0,\ if\ x_j = 0 \\ x_i \in [1,\ x_j],\ otherwise \end{cases}$
Named :Union Description :Groups the tuples of two structures into a single one. Noted :$\cup$	$\forall\ s_j = (C,\ D_j),\ s_k = (C,\ D_k)$ $\Rightarrow\ s_j \cup s_k = (C,\ D_j \cup D_k)$ Resulted tokens number : $x_i \in [Max(x_j, x_k),\ x_j + x_k]$
Named :Intersection Description :Retrieves the common tuples of two structures. Noted :$\cap$	$\forall s_j = (C,\ D_j),\ s_k = (C,\ D_k)$ $s_j \cap s_k = (C,\ D_j \cap D_k)$ Resulted tokens number : $x_i \in [0,\ Min(x_j, x_k)]$
Named :Division Description :Allows to get a data structure tuples that are associated with all tuples of another structure. Noted :$\div$	$\forall s_j = (C_j,\ D_j),\ s_k = (C_k,\ D_k)$ Where :$C_j = (c_1 \dots c_{n_j})$, $C_k = (c_1, \dots c_{m_j})$ If $n_j > m_j$ then : $\begin{cases} s_i = s_j \div s_k = (C_i,\ D_i) \\ C_i = (c_{m_j+1},\ c_{m_j+2} \dots c_{n_j}) \\ \forall\ q \in D_i,\ D_k \times q \in D_j \end{cases}$ Resulted tokens number : $x_i \in [0,\ E(x_j/x_k)]$
Named :Substitution Description :Changes a structure attribute name. Noted :$\boxtimes$	$\forall s_j = (C_j,\ D_j),\ s_i = \boxtimes(c_{j_k},\ c,\ s_j)$ $\Leftrightarrow s_i = ((c_{j_1} \dots c_{j_k - 1}, c, c_{j_k + 1}$ $\dots c_{j_n}),\ D_j)$ Resulted tokens number :$x_i = x_j$
Named :Permutation Description :Allows to permute two columns in a data structure. Noted : $\overset{\frown}{\smile}$	$\forall s_i = (C_i,\ D_i), s_j = \overset{\frown}{\smile}(s_i,\ k,\ l)$ $\Leftrightarrow \begin{cases} k,\ l \in \{1,2 \dots n\} \\ k < l \\ C_j = (c_{i_1} \dots c_{i_{k-1}}, c_{i_l}, c_{i_{k+1}} \\ \dots c_{i_{l-1}}, c_{i_k}, c_{i_{l+1}} \dots c_{i_n}) \\ D_j = \{(d_{1i_1} \dots d_{1i_{k-1}}, d_{1i_l}, \\ d_{1i_{k+1}} \dots d_{1i_{l-1}}, d_{1i_k}, d_{1i_{l+1}} \\ \dots d_{1i_n}), (d_{mi_1} \dots d_{mi_{k-1}}, \\ d_{mi_l}, d_{mi_{k+1}} \dots d_{mi_{l-1}}, d_{mi_k}, \\ d_{mi_{l+1}} \dots d_{mi_n})\} \end{cases}$ Resulted tokens number :$x_i = x_j$
Named :Extension Description :Extends a structure scheme by adding a attribute c =(n,t) and applying a function f. Noted :$\dot{-}$	$\forall s_j = (C_j,\ D_j),\ s_i = \dot{-}(s_j,\ c,\ f)$ $\Leftrightarrow s_i = ((c_{j_1}, c_{j_2} \dots c_{j_n}, c), \{(d_{j_{1_1}},$ $d_{j_{1_2}} \dots d_{j_{1_n}},\ f(d_{j_{1_1}}, d_{j_{1_2}} \dots d_{j_{1_n}},$ $D_j)) \dots (d_{j_{m_1}}, d_{j_{m_2}} \dots d_{j_{m_n}}, f(d_{j_{m_1}},$ $d_{j_{m_2}} \dots d_{j_{m_n}},\ D_j))\})$ Resulted tokens number :$x_i = x_j$
Named :Add_Tuple Description :Add a tuple of data d in a data structure. Noted :$+$	$\forall s_j = (C_j,\ D_j),\ D_k = (d_{k1}, d_{k2} \dots d_{kn}),$ $s_i = +(s_j,\ D_k)$ $\Leftrightarrow s_i = ((c_{j_1} \dots c_{j_n}), \{(d_{j_{1_1}}, d_{j_{1_2}} \dots$ $d_{j_{1_n}}) \dots (d_{j_{m_1}}, d_{j_{m_2}} \dots d_{j_{m_n}}), (d_{k1},$ $d_{k2} \dots d_{kn})\})$ Resulted tokens number :$x_i = x_j + 1$
Named :Copy Description :Makes n copies of a data structure. Noted :t_{copy}	$\forall s_i = (C_j,\ D_j),$ $t_{copy}(s_i,\ n) = \{S_{j_1}, S_{j_2} \dots S_{j_k}\},$ where $k \in \{1,2 \dots n\}$ and $s_j = s_i$. Resulted tokens number :$x_j = x_i$

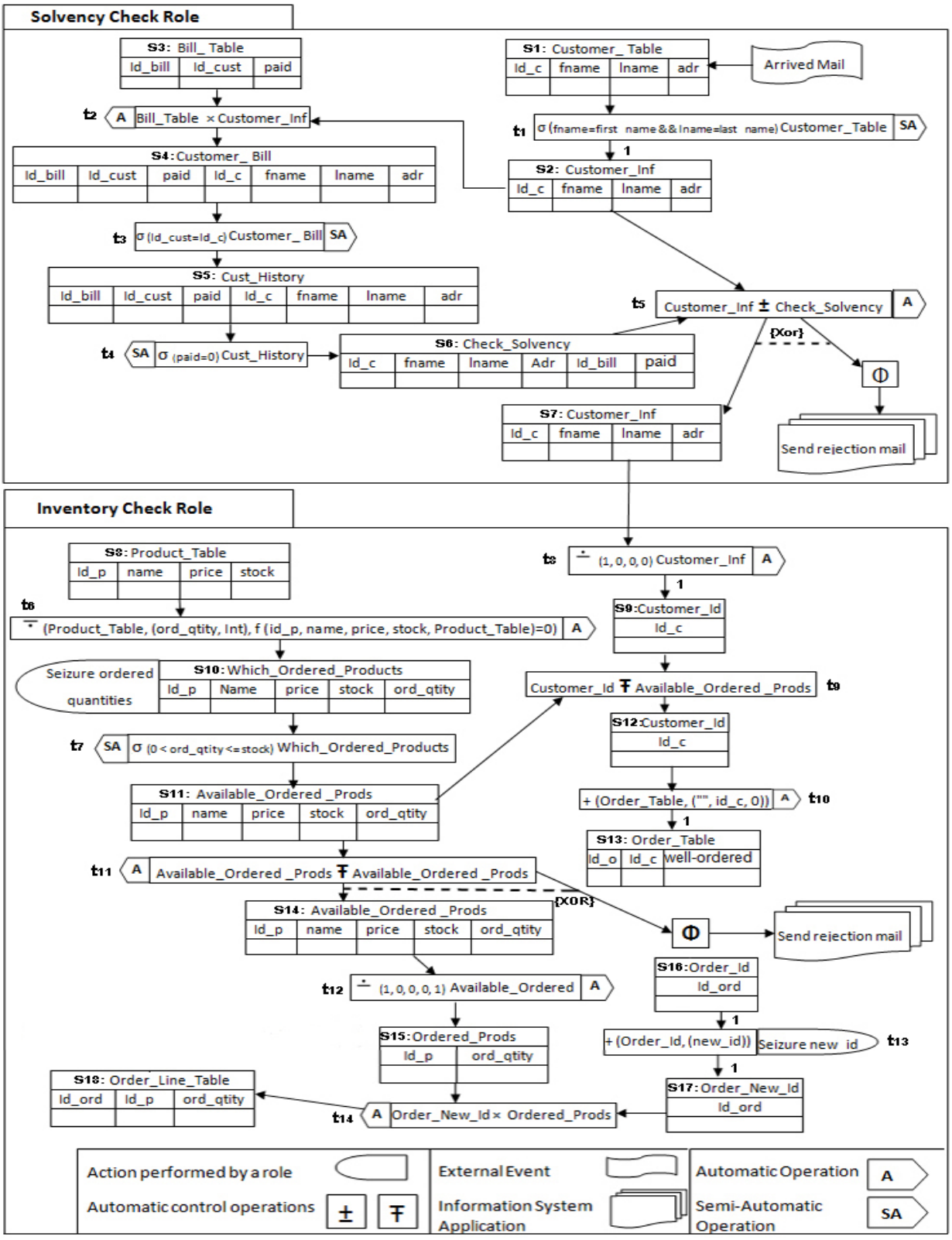

Figure 5. Orders management workflow