A Formal Approach for Business Processes Description, Customization and Automation

Nahla Haddar, Mohamed Tmar, and Anis Jedidi

Abstract—The importance of data in business processes has increased progressively in recent years motivated by the need for executable workflow models according to the process control rules. In this paper we describe an approach for data-driven workflow process modeling. Based on structured tokens Petri Net, the resulting workflow model can represent both control flow and shared data in the workflow process. Besides, to prove our modeling approach, we implement the Opus workflow management system. It provides a graphical editor to model a customize workflow process, and a workflow engine to automatically execute the modeled workflow process.

Index Terms—Data-driven approach, petri net, structured tokens, workflow process definition.

I. INTRODUCTION

At the beginning of this century, workflow management concentrated on the design and documentation of business process. Therefore it focused on the dependencies between tasks and their sequencing, whereas data and resources (person realizing the task or application interacting with the task) played a very minor role. Many new approaches have been introduced, e.g., Petri Nets [1], Business Process Modeling Notation (BPMN) [2], Business Process Execution Language (BPEL) [3], etc, but only a few of them are of ongoing interest in modeling the exchanged data flow in the business process. The business process management requires explicit statement of data objects which express pre and post conditions of consuming data during process execution. Accordingly, the data-driven approaches appeared during the last years. In this paper, we present a formal approach inspired from the data-driven approach and the Petri Net formalism to model workflow process. Besides, the proposed model must facilitate the possibility to change a sub-process model without damaging the whole performed process model.

The rest of the paper is organized as follows: we illustrate the related works in Section II, and then we elucidate our approach for workflow modeling in Section III. In Section IV, we illustrate the possible information flows routing. We demonstrate our approach by illustrating an example of workflow model in Section V. In Section VI, we present our workflow management system Opus. Section VII concludes.

II. RELATED WORK

The activity-driven approaches (e.g., BPEL [3], BPMN [2], Petri Nets [1], and UML Activity Diagrams [4]) mainly focus on the representation of the control flow. Some approaches partly allow the data objects modeling in process structures. For example, the UML 2.0 Activity Diagrams allow multiple instantiation of activities based on simple data structures (set, list) [5] and in BPMN, the data are represented by the artifacts (annotation, data object,...) but the explicit statement of data objects state or structure are not supported [2].

At the beginning of this century, many new approaches have emerged which shifted their focus to combination of data flow and control flow. An emerging approach uses artifacts and Petri Nets, that combine data and process is the Business Artifacts (BA) [6]-[7]. According to the formalization developed in [7], the BA approach focuses on solving decision problems, related to reachability, avoiding dead-ends and redundancy [6] but it does not provide a graphical notation for process modeling.

Another formal approach basis on Petri Nets model is the CorePro Framework [8]. It enables to model the data-driven specification and then, to create automatically the process structures based on given data structures in the model level. However, it has skipped to retain the object states which have already been activated before the execution. In fact, in addition to structural changes, it has also to consider state changes.

The approaches mentioned above focus on the data routing and data managed by the process, but they consider activities as black-boxes in which application data is managed by invoked application components. However, an activity becomes activated during the process execution depends on the state of other activities and on its resulting data. These processes are, in particular, processed according to user decisions and knowledge-intensive [9]. This is why process instances have to be synchronized at a higher-level of abstraction to reflect the preferred work practice.

III. WORKFLOW MODELING USING STRUCTURED TOKENS PETRI NET

We have inspired from the Petri Nets to propose a new workflow modeling approach that incorporates a workflow method leading to a workflow process model. The workflow model must be able to manage all the data handled by the work procedures in the organization. So, we use the notion of data-driven process structures to model the process, which is described by respective data structures. We define a data structure as a pair $s = (C, D)$ where $C$ is a list of fields and $D$ is
structures through the definition of an operations graph.

Each field $ci$ is defined by a name $ni$ and a type $ti$, such as:

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

2) $\forall i, j, dij \equiv tj$: values must be compatible with the types of fields.

Formally, we define a workflow by $W = (R, G, I)$ where:

1) $R$ denotes a set of cooperating roles in the workflow,
2) $G$ denotes an operations graph,
3) $I$ denotes the information system, providing data to the workflow.

According to this definition, we identify all the human resources $R$ to roles rather than persons (a user or a group of users can be assigned to one or more roles). A role is to be defined by a homogeneous set of tasks. Besides, we consider a workflow task as the transformation of an input information to another output. Formally, a role can be defined by a finite set of data structures $S$ which are manipulated by the role itself. So, we define a workflow as a set of roles $R$ cooperating by exchanging information as a set of data structures.

Each role can product the content of its output data structures through the definition of an operations graph $g \in G$ expressing his usually performed tasks.

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 (See Table I). As illustrated in Table I, we redefine the relational algebra operations (Selection, Projection, Union, Difference, Intersection, Division, and Inner product) in a formal way. And to keep compatibility and equivalence between the structures fields assigned to operations as Union, Difference, Intersection, and Division, we defined the Permutation and Substitution operations.

Furthermore, users of the workflow may need to add fields in a structure where values are not generated through applying a function. To do so, we suggest an Extension operation for the structure schema. And finally, to insert data tuples in a structure, we define the Alimentation operation. All the operations consume data structure tuples and produce other tuples.

The difference between a Petri Net model and our operations graph is that we can not explicitly assign the cardinality of the reachability to an operation.

In fact, each transition consumes an undefined number of data tuples (tokens) and produces a number belonging to a well determined range, depending on its processing (See Table I).

IV. INFORMATION FLOWS ROUTING

As a Petri Net using relational algebra operations as transitions, our workflow model can easily express sequential and parallel routing. However, conditional routing needs to be defined by other operations, so, we define the control operations. According to the data structure content, the system should be aware to decide whether to continue the information flows routing or not. Consider $sj$ is the controlled data structure and $si$ is the data structure expected by the next transition in the model. Then the structure $sj$ is to be controlled by one of the control operations which are defined as follows:

Control operation 1 noted $\pm$:

$$s_i \pm s_j = \begin{cases} s_i & \text{if } s_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

Control operation 2 noted $\mp$:

$$s_i \mp s_j = \begin{cases} \emptyset & \text{if } s_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

Fig. 1 illustrates an example of conditional routing which checks the solvency of a customer using the $\text{Customer History}$ structure.

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 letter is sent to the customer, or the order is approved and then sent to shipping and billing.

In this paper, we restrict our example to the solvency check process and the inventory check process. Considering the information system tables as data structures, as shown in Fig. 2, the Solvency Check Role receives the structure $\text{Bill Table}$ and $\text{Customer Table}$. He uses $\text{Customer Table}$ able to select the information about the current customer. The resulting structure $\text{Customer Inf}$ and the $\text{Bill Table}$ structure are to be used to get the current customer history in order to check the customer solvency on bill payment.

According to the solvency result, the $\text{Customer Inf}$ is sent to the Inventory Check Role to launch his sub-process execution.

Otherwise, the Solvency Check Role has to send a rejection mail to the customer. As for the Inventory Check Role, he receives $\text{Product Table}$ and $\text{Order Table}$ from the information system. $\text{Product Table}$ is used to select the available products for the current order. If the ordered products are available, the $\text{Inventory Check Role}$ inserts a
new order in the *Order_Table*. Then, he types the new order identifier to produce the *Order_Lines* structure which is saved in the *Order_Line_Table* of the information system. Otherwise, the *Inventory Check Role* has to send a rejection mail to the customer.

VI. IMPLEMENTATION OF THE WORKFLOW MANAGEMENT SYSTEM OPUS

The Opus workflow system consists of a number of components including a Petri Net editor and a workflow engine. Workflow specifications can be designed using the Opus editor and then, the Opus engine follows the workflow model definition and interprets automatically the code executing the workflow. Indeed, it invites each role to perform its tasks according to its feasibility and urgency. To facilitate the role tasks, we developed some tools, e.g., Insert operation (it inserts a data structure tuples in the information system table, so it can replace the Alimentation operation when the alimented structure is a table from the information system), the ImportId operation (it imports the tuple identifier of the last tuple inserted in a definite table) and the Import operation (it imports a table tuples to a definite data structure belonging to the workflow process). To perform these operations, Opus system is equipped with a matching tool, which uses the Substitution and the Permutation operations, to specify for each field of the structure its suitable field in the table. It can be also necessary to the workflow engine to execute operations which requires two identical data structure schemas, such as Union operation.

A. Orders Management Workflow Example Modeling Using Opus Editor

The first step in the workflow modeling is to create role profiles by a default role (Workflow Administrator) created by the system. Then each role should model its individual workflow. The modeling phase starts with defining the exchanged information flows (data structures) between roles, the information system database or with other applications from the information system (see Fig. 3. *Solvency Check Role* is presented by *Role1*, *Inventory Check Role* is presented by *Role2*, the information system is presented by *SI* and the *Outlook* application is presented by *App*). The next step in modeling phase consists in defining the data schema for each exchanged data structure. Then, each role can turn to define its sub-process model. In the studied example, the workflow is defined by two sub-processes model related to *Role 1* (see Fig. 4.) and *Role 2* (see Fig. 5.) and connected by the data structures exchanged between the two roles.
B. Orders Management Workflow Example Processing Using Opus Engine

Until workflow modeling is achieved, the Workflow Administrator still has to connect the workflow with the information system. For us, we consider the information system as a relational database. During the processing of the workflow, the workflow engine assigns instances of data structures to roles by following the information flows and executing the whole workflow process definition conceived in the modeling phase. So, in this example, the Opus workflow engine finds the Role1 individual workflow, and starts by instantiating its input structures (Bill_Table and Customer_Table). To create the tuples of input structures, the system allows the role to manually type the data or to import them from the appropriate table in the information system. If all the input structures are instantiated, the Opus engine will automatically execute the sub-process related to the connected role and will present the results in order to be validated and sent to the next role. If a transition requires a role intervention (e.g., performing a matching), the concerned role will be notified by the system. Resuming our example, Role 1 imports data to Customer_Table and validates its content to launch the processing of its sub-process as shown in Fig. 6. Fig. 7, illustrates the execution of the Role1 individual workflow. In the same way, the Opus workflow engine automatically executes the Role 2 individual workflow, once Role 2 instantiates the structure imported from the information system (Products_Table and Order_Table) and validates its contents.

VII. CONCLUSION

In this paper, we have presented an approach that leads to a process model useful for the definition and processing of workflows. The approach is modular in a sense that each role can model its individual workflow, which facilitates any eventual performed updates on the workflow. However, we intend to extend our approach to verify the correctness of the workflow process, test if it is behaving as expected, and evaluate its ability to meet the requirements of processing time, service levels and resource utilization.

APPENDIX

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formal definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Named: Selection</td>
<td>( \forall s_j = (C_j, D_j) ), ( s_j = \sigma_{p} s_j )</td>
</tr>
<tr>
<td>Description:</td>
<td>( (C_j, D_j) ) and ( { e } )</td>
</tr>
<tr>
<td>Noted:</td>
<td>( p(e) )</td>
</tr>
<tr>
<td>Resulted tokens number:</td>
<td>( x \in [0, x_i] )</td>
</tr>
<tr>
<td>Where:</td>
<td>( \exists c \in {0,1}^n )</td>
</tr>
<tr>
<td>Named: Projection</td>
<td>( s_j = (C_j, D_j), (b_1 ... b_n) \in {0,1}^n )</td>
</tr>
<tr>
<td>Description:</td>
<td>( \exists x_1 \in [0, x_i], x_1 \in {1, x_j}, \text{otherwise} )</td>
</tr>
<tr>
<td>Noted:</td>
<td>( s_j = (C_j, D_j), x_1 = \text{Validated} )</td>
</tr>
<tr>
<td>Named: Inner Product</td>
<td>( \forall s_j = (C_j, D_j), s_j = (C_j, D_j) )</td>
</tr>
<tr>
<td>Description:</td>
<td>( \forall s_j = (C_j, D_j), s_j = (C_j, D_j) )</td>
</tr>
<tr>
<td>Noted:</td>
<td>( \forall s_j = (C_j, D_j), s_j = (C_j, D_j) )</td>
</tr>
<tr>
<td>Named: Difference</td>
<td>( s_j = (C_j, D_j), D_j )</td>
</tr>
<tr>
<td>Description:</td>
<td>( \forall s_j = (C_j, D_j), s_j = (C_j, D_j) )</td>
</tr>
<tr>
<td>Noted:</td>
<td>( \forall s_j = (C_j, D_j), s_j = (C_j, D_j) )</td>
</tr>
<tr>
<td>Named: Intersection</td>
<td>( s_j \cap s_k = (C_j, D_j) )</td>
</tr>
<tr>
<td>Description:</td>
<td>( s_j \cap s_k = (C_j, D_j) )</td>
</tr>
</tbody>
</table>

TABLE I OPERATIONS DEFINITION

\[
\begin{align*}
&\forall s_j = (C_j, D_j), s_j = (C_j, D_j) \\
&\forall s_j = (C_j, D_j), s_j = (C_j, D_j) \\
&\forall s_j = (C_j, D_j), s_j = (C_j, D_j) \\
&\forall s_j = (C_j, D_j), s_j = (C_j, D_j) \end{align*}
\]
<table>
<thead>
<tr>
<th>Noted: ⊗</th>
<th>$x_i \in {0, \text{Min}(x_y, x_z)}$</th>
</tr>
</thead>
</table>
| **Named: Division**<br>Description: Allows getting a data structure tuples that are associated with all tuples of another structure. | $\forall s = (C_i, D_j), s_r = (C_k, D_l)$<br>Where: $C_i = (c_1, \ldots, c_{n_i}), C_k = (c_1, \ldots, c_{n_k})$
If $n_i > n_k$ then: $s_i = s_k \iff (C_i, D_j)$<br>$C_i = (c_{m_1}, c_{m_2}, \ldots, c_{m_n})$
$\forall q \in D_l, q \neq x \in D_j$
Resulted tokens number: $x_i = 0, \text{E}(x, y, z)$ |
| **Named: Substitution**<br>Description: Changes a structure field name. | $s_i = (C_i, D_j), s_r = (c_j, c, s)$ | $s_i = ((c_i), (c_{j,k}), (c_{j,k+1}), (c_{j,k+2}), \ldots, (c_{j,m}), D_j)$<br>Resulted tokens number: $x_i = x_j$ |
| **Named: Permutation**<br>Description: Allows to permute two columns in a data structure. | $s_i = (C_i, D_j)$<br>$s_j \sim (s_k, k, l)$<br>$k, l \in \{1, 2, \ldots, n\}$<br>$k < l$
$\Leftrightarrow D_j = \{ (d_{1,k}, d_{1,l}, d_{1,k+1}, \ldots, d_{1,l}, d_{2,k}, d_{2,l}, \ldots, d_{m,k}, d_{m,l}) \}$<br>$s_j = (c_j), D_j$ | $\text{Resulted tokens number: } x_i = x_j$ |
| **Named: Add-Tuple**<br>Description: Add a tuple of data in a data structure. | $s_i = (C_i, D_j), D_k = \{ d_{k,1}, d_{k,2}, \ldots, d_{k,n} \}$, $s_j = + (s_j, D_k)$ | $s_i = ((c_{j,k}), c_{j,l}), \{ (d_{j,1}, d_{j,2}, \ldots, d_{j,n}) \}$<br>$\text{Resulted tokens number: } x_i = x_j + l$ |
| **Named: Extension**<br>Description: Extends a structure schema by adding a field $c = (n, t)$ and applying a function $f$. | $s_j = (C_j, D_j)$, $s_i = \tau (s_j, c, f)$<br>$s_i = ((c_{j,1}, c_{j,2}, \ldots, c_{j,n}), (d_{j,1}, d_{j,2}, \ldots, d_{j,n}), f(d_{j,1}, d_{j,2}, \ldots, d_{j,n}, D_j), \ldots, (d_{m,1}, d_{m,2}, \ldots, d_{mn}, f(d_{m,1}, d_{m,2}, \ldots, d_{mn}, D_j))$ | $\text{Resulted tokens number: } x_i = x_j$ |

**REFERENCES**


**Nahla Haddar** was born on September 6, 1985 in SFax, Tunisia. She received Master degree in new technology and computer system dedicated from the National School of Engineers of SFax in January 2011. She is working towards her Ph.D at the Faculty of Economic and Management of SFax, and a researcher member at Multimedia Information Systems & Advanced Computing Laboratory (MIRACL) of SFax University. Her research interests include modeling, verification techniques and automation of business process

**Mohamed Tnar** was born in Sfax (Tunisia) on January 11, 1975. He received his Master degree in data processing and telecommunication from Paul Sabatier University, Toulouse III, France, 1998, received his Ph.D degree in Computer Science from Paul Sabatier University, Toulouse III in July 2002. He is currently a Professor at the Higher Institute of Computer Science and Multimedia of SFax. His research interests are in many areas of information retrieval, including retrieval models, representation, Web search, query processing, medical retrieval, and XML retrieval, and Business Process Management. Dr. Tnar is also a member at MIRACL and he has served on numerous program committees, and has been involved in the organization of many workshops and conferences.

**Anis Jedidi** was born in Sfax (Tunisia) December 01, 1976. He received his Master degree in data processing and telecommunication from Paul Sabatier University, Toulouse III, French, 2000, received his Ph.D in Informatics from the Paul Sabatier University, Toulouse III in July 2005. He is currently a Professor at the Higher Institute of Computer Science and Multimedia of SFax. His research interests include semi-structured document modeling, multimedia document design, semantic Web, annotation and querying. He interests also on extension of XML query language for multimedia document and presentation and pervasive information system. Dr. Jedidi is also a member at MIRACL and he works now, with Ph.D. students on the annotation of semantics features in multimedia and Web resources.