Activity-Based Workflow Mining: A theoretical framework

Carlos Fernandez
TSB - Health and Wellbeing Technologies group, UPV Spain
carferll@eln.upv.es

Jose Miguel Benedi
DSIC UPV Spain
jmbenedi@dsic.upv.es

Abstract

This paper presents a Workflow Mining new paradigm. This paradigm keeps in mind on one side the processes,and on the other hand the results given by these processes themselves. This is a contrast between the Event-Based Workflow Mining and the one it is presented in this paper. Finite Automata are a good idea to represent this paradigm to take profit of the theoretical advantages of the Regular Languages. Also, time representation in a workflow is another serious problem to be solved. In this paper it is also presented a new formal framework(TPA)based on Finite Automata capable to represent time, keeping the complexity of Regular Languages level.

1 Introduction

The process standardization traditionally was basically based on the hand-made processes creation who explained the process in natural language. Nevertheless, the first problems took not too long to appear, those manual processes became larger and larger making more difficult to deal with them. This problematic made grow up the guidance difficulty. With the arrival of new technologies the idea of the processes automation gains more and more acceptance. The automatic guiding of the processes could deal perfectly with the standardization process problems. With this, a new research branch was born, known by *Workflow Technology*[Coa99].In addition, in [vdA96] Aalst informs about the need of workflow engines formalization to increase its expressiveness and standardization.

The great amount of languages and formal specifications which describe workflows [vdA96] [GHS95] [KtHB00] [vdAH02] are growing in an exponential way, promoted by the commercial demand of those applications. Nevertheless, in spite of the wide need to obtain effective and efficient Workflows, a few formal workflow management studies have been relatively realized.

A very good study, about the languages and specification representation capability is presented in[vdABtHK03]. In [vdABtHK03] Aalst proposes a set of patterns that every workflow should fulfill to cover all the needs that are indispensable in a workflow process. Besides, in this paper is shown a comparative study of the expressivity capability of the most important workflow tools available in market. Any commercial tool fulfill the workflow patterns as it is concluded in this paper.

Usually the processes that are susceptible to be automated, are very complex. The complexity of those processes are directly proportional to the formalized model complexity. This problem can make the difference between the system's success or failure because, actually, in spite of the great amount of languages and paradigms that try to make the creation of workflows more standard by hand using formal paradigms is far to be trivial. In practice, we can use learning techniques to infer the workflow, helping the system designer to create workflows using examples. Continuing, it is important to emphasize the need of utilization of formalized frameworks that can guarantee the consistency of the designed model. The idea of automatic learning on workflows is known as *Workflow Mining*

The Workflow Mining or Process Mining is a technique that allows to infer workflows using as input collected samples from execution time processes. This is made in order to offer information to the workflow designer on execution time [vdAvDH$^+$03]. The main challenge of a workflow mining system is to infer good models with the least information possible. Workflow Mining is a powerful tool to processes designers, that can help them with a better system's approach with a real view of the system's execution.

The studied models of Workflow Mining [CW98] [WvdA] [vdAWM03] are based in the utilization of transactional logs as samples for workflows inferences. Using different techniques, those models use systems as ERP, CRM or even other general systems of workflow management as input samples, to infer models that explain the whole system. Those samples are based on logs of those systems that means they are based on a set of events, it's to say, every sample is a fact at a point of time. This approach is known as Event-Based Workflow Mining[vdAWM03].

The Event-Based Workflow Mining assumes that the system that will be inferred has capability to record events that have referred to a well-known system task, ordered in time, and which can be separated into individual processes in each case. In fact, it is not necessary to have a previous system that collects the information, it is possible to apply Workflow Mining to a log on execution time.

The Event-Based Workflow Mining problems are a very good approach to the enterprise's reality. In fact, most of companies can use Workflow Mining to infer their processes using the logs of their software systems with no changes. In addition, due to the easy way to get big amounts of samples for the inference algorithm, the learning success is more probable in a world where the lack of data is the normal problem.

Nevertheless, this approach have difficulties to guide the process because it does not take into account the 'why' of the changes in the process. This information in some special cases is crucial, for example, in case of medical care paths usually the change of state in a processes is due to the task's result (to get blood preasure) and depending on the result of this task (Hypotense, Normotense or Hypertense),it goes from one state or another. The event-based Workflow Mining approach have problems to deal with those type of cases. Completing this approach, it must be emphasized that is very difficult to deal with time. For example, if there is a task with the main purpose of waiting some time (wait 5 hours), it is difficult to represent it.

The main objective of this paper is to present a new workflow mining paradigm that can deal with the problems that Event-Based Workflow Mining can not solve. In this way, at section 2 this new paradigm, called Activity-Based Workflow Mining is presented. The Activity-Based Workflow mining main characteristic is that take into account not only the tasks, but also the results of those tasks incorporating this knowledge into the model, taking into account time. Nevertheless, the principal problem that Activity-Based Workflow Mining Model has is the need a formal framework which model formally this Workflow Mining case. In this way, a new formal framework in order to deal with the paradigm is presented on section 3,that are capable to represent the tasks with its results, time taking into account. In addition, the formal framework presented must be capable to represent the 20 Aalst patterns in order to be sure about theframework representation capablility. Finally, section 5 concludes de paper.

2. Activity-Based Workflow mining

As it has been shown before, the Event-Based Workflow Mining paradigm is not able to deal properly with the information appointed by the result of the monitored tasks. In order to add this information into the learning model, a new paradigm called Activity-Based Workflow Mining is presented. In this paradigm, the event concept is occupied by the activity concept. The activity concept take into account not only the task of the processes realized, but also the task's result in order to define the following step. The main purpose of this specific kind of Workflow Mining is to infer workflows that are capable to guide the processes in order to standardize up to the maximum the real process. In addition, the Activity-Based Workflow Mining algorithms must be capable to infer and represent time, in order to know not only *what* must a process do, but also *when* the process must be realized.

In an analog way to the Event-Based Workflow Mining, the Activity-Based Workflow Mining assumes that the inferred system has capability to record activities that has referred to a system's well-known task, ordered in time, which can be separated into individual processes in each case. Besides, the Activity-Based one can get its results attached to the task.

There are several applications that can take profit from the Activity-Based Workflow Mining systems. Following section enumerates some of them:

1. Medical Care Plans Systems:

   The Evidence Based Medicine (EBM) [SRGH05] is based on the use of care plans to standardize health processes due to certain diagnosis the patients must suffer. These care plans base their decisions on the indicators resultant of each activity the patient bear, and, based on these indicators it is decided to follow, finish, or change the plan. In this case, it is very important the time model, because in some cases it is needed some wait to do some activities, for example, one day wait is necessary in order to find out if a medication takes effect or not. Workflow Mining models can help these systems, aiding the doctors to design workflows from the daily practice standardization. As we know, the Event-Based Workflow Mining system, has problems to define time and the activities results, nevertheless,the Activity-Based Workflow Mining model can deal with this.

2. Human Behavior Modeling Systems:

   Nowadays, the Human Behavior Modeling systems are deeply studied, in order to model personal environments on Ambient Intelligence Systems. These systems study the human behavior, to be able to find the behavioral patterns that allow to know more about the human being under study under special circumstances. It is hoped from this discipline that it allows to help the design of better adaptable user's interfaces, help elderly people, alzheimer illness patients etc... In this case, creating hand-made workflows for each person is not effective. Workflow Mining models can help these systems making an inference on personal workflows for each people. As we know, the Event-Based Workflow Mining system, has problems to define this models, where time(e.g. control the bedridden time for elderly people) and the activities results (e.g. take shower's water temperature for elderly people)is crucial. Nevertheless, the Activities-Based Workflow Mining model can deal with this.

3. Traffic Control Systems: The main challenge in Traffic Control Systems is to improve the vehicular traffic flow, providing information and controlling and managing traffic. That is the reason why studies are made continuously in urban environments in order to allow to define, by hand, the regulation model needed to be implemented in the traffic lights for greater efficiency. The Event-Based Workflow Mining systems utilization is not able to infer the system properly, since it would be based on defining workflow the traffic lights follow avoiding the traffic flow indicator that would be studied by an Activity-Based system.

As it is known, Activity-Based Workflow Mining has its advantages in some specific cases representation about Event-Based Workflow Mining systems. On the other hand, at it was expected, the Activity-Based Workflow Mining it is more difficult to solve than Event-Based Workflow Mining problem. Like in Event-Based Workflow Mining, Activity-Based problems can take profit from the system's log to make inference. Nevertheless, most of logs actually used, do not have enough information for the Activity-Based problems. In practice, it is necessary to modify systems to infer the models. This makes more difficult to get data to infer models, having the learning classical problem of lack of data.

Concluding, it is crucial to get a formal framework capable to deal with Activity-Based Workflow Mining problems. It must be capable to represent activities, its results and the activity time, making possible the model guidance. In addition, it is important to create a theoretical consistent framework in order to allow to guarantee the model inferred consistency.

In following section,a formal framework that can deal with the problem is presented.

3 Timed Parallel Automatons (TPA)

In order to build a formal framework capable to deal with Activity-Based Workflow Mining problems.In practice, due to the characteristics of the workflows, it is better to solve Workflow Mining problems using syntactical instead of geometrical approaches, as can be seen in [CW98]. Due to this reason, only syntactical frameworks will be studied.

In current workflow modeling theories, Petri Nets[Mur89]has taken the main role for lost of reasons[vdA96]. The Petri Nets are capable to allow representing a wide quantity of workflows, to deal with multithreading processes and have lots of analysis techniques that can be applied to the system... What makes out of them a powerful framework to solve workflow mining problems. Nevertheless, although the Petri Nets are very expressive, very simple and powerful, its complexity is potentially very high, what means that the languages that can be recognized by the Petri Nets are much more complex that the regular languages class[JEM99], dificulting the inference process. Adding to this problem, it is difficult to represent time in this kind of nets.

On the other hand, in order to preserve the framework complexity Deterministic Finite Automata Models are other approach to take into account.Nevertheless, Those models are not able to represent models with several states at the same time. This problem has been treated and solved in many papers [Gra98] [GMA95] [MY01] [SP94] defining an improvement of the Determinist Finite Automata that presents an automaton with capability to execute parallel activities, keeping the complexity at regular level. In spite of this, those systems are no capable to represent time.

Although, the time model in workflows can be very useful, in the models defined previously it is not taken into account. There are some studies that try to take into account time for State Finite Machines [AM04] [AD94] [SVD01] [D'A] and in Petri Nets[Ram74]. There are two main ideas to represent time: dense time modeling, where it uses a real number to represent time, or the discreet time modeling where it models time using discreet values.

From the Activity-Based Workflow Mining problem point of view, dense time modeling, would directly affect the problems complexity and will avoid the system to maintain a regular complexity. Nevertheless, the use of discreet time modeling with a finite number of labels, built-in the automaton alphabet, would not affect to the language accepted by the model, therefore, it would continue being regular.

Due to the necessity to create a new framework in order to deal with Activity-Based Workflow Mining paradigm, a new formal framework able to define a parallel automata and capable to represent the time maintaining the equivalence to a Determinist Finite Automata is defined following.

Definition 3.1 *A Clock C^i is a machine so once passed a time interval i from last time it was reset, it generates a symbol $t \in T$*

Examples:

C^{10} *this Clock generates the symbol* t^{10} *10 seconds after*

C^0 *this Clock generates the symbol* t^0 *instantaneously*

C^∞ *this Clock never generates any symbol*

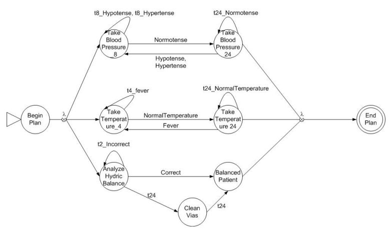

Figure 1. Medical care plan TPA example

Definition 3.2 *A Timed Parallel Automaton(TPA) is a tuple* A $=\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$ *where:*

C is a finite set of Clocks,

P is a finite set of Processes,

$N \subseteq P \times C^+$ *is a finite set of Nodes,*

$Q \subseteq N^+$ *is a finite set of states,*

T is a finite set of time labels that can be generated by the Clock set C,

Φ *is finite set of task results,*

$\Sigma \subseteq T \cup \Phi^+ \cup T \times \Phi^+ \cup \{\lambda\}$ *is the finite input alphabet,*

$\gamma : N^+ \times \Sigma \rightarrow N^+$ *is the Node Transition function,*

$\delta : Q \times \Sigma \rightarrow Q$ *is the state transition function,*

$q_0 \in Q$ *is the initial state,*

$F \subseteq N$ *is the set of final states.*

The defined TPA is based on the definition of PFA, incorporating the Clock concept. Each node contains a Clock or a set of Clocks inside, that begins counting when the state is reached, and emit his associated symbol when the time has happened.

The set of labels that creates the set of Clocks defined denominated T. On the other hand the set of indicators or symbols is denominated ϕ. These two sets are going to form the alphabet σ, that will contain the set T, to label the time only dependent transitions, the ϕ positive closure of indicators only dependent label transitions, and the product of the set T by the indicators sets that allows to label the time and indicators dependent transitions.

In the figure 1,it is shown a TPA use example in a medical care plans process. In this process temperature, blood pressure and hydric balance of an entered patient is controlled. During plan initialization, activities are parallely executed to take temperature, blood pressure, and to analyze the hydric balance. In blood pressure taking case, according to the figure, the blood pressure must be taken every eight hours, in case from hypertense and hypotense. In case of normotense the blood pressure must be taken every twenty-four hours. In a similar way, in the temperature taking case, if the patient has fever, temperature must be taken every four hours, and temperature must be taken every twenty-four hours if the patient has no fever. In both cases, if the patient disease gets worse, the state changes to the more frequently making measures state. Finally, in case of hydric balance, the hydric balance is analyzed every two hours until it returns into the correct range, case where the patient would be balanced. If during twenty-four hours later he/she continues without correcting itself, the patient will be probed and twenty-four hours later he/she would be changed automatically to the balanced state. An automata formal representation is presented following.[1]

A $=\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$

C = $\{c^\infty, c^2, c^4, c^8, c^{24}\}$,

P = {BeginPlan, TakeTension8, TakeTension24, TakeTemperature4, TakeTemperature24, AnalizeHydricBalance, Probe, BalancedPatient, EndPlan},

N = $\{n_0 : [BeginPlan, c^\infty]$,

$n_1 : [TakeTension8, c^8]$,
$n_2 : [TakeTension24, c^{24}]$,
$n_3 : [TakeTemperature4, c^4]$,
$n_4 : [TakeTemperature24, c^{24}]$,
$n_5 : [AnalizeHydricBalance, c^2c^{24}]$,
$n_6 : [Probe, c^{24}]$,
$n_7 : [BalancedPatient, c^\infty]$,
$n_8 : [EndPlan, c^\infty]\}$

Q = $\{q_0 : n_0$, $q_{135} : n_1n_3n_5$, $q_{235} : n_2n_3n_5$, $q_{145} : n_1n_4n_5$, $q_{136} : n_1n_3n_6$, $q_{137} : n_1n_3n_7$, $q_{245} : n_2n_4n_5$, $q_{236} : n_2n_3n_6$, $q_{237} : n_2n_3n_7$, $q_{146} : n_1n_4n_6$, $q_{147} : n_1n_4n_7$, $q_{246} : n_2n_4n_6$, $q_{247} : n_2n_4n_7$,$q_8 : n_8\}$,

T = $\{t^2$, t^4,t^8,$t^{24}\}$,

Φ = {Hypotense, Hypertense, Normotense, Fever, NormalTemperature, Ok, NotOk},

Σ = {Hypotense, Hypertense, Normotense, Fever, NormalTemperature, Ok, NotOk, t^2, t^4, t^8, t^{24}, t^8Hypotense, t^8Hypertense, t^4Fever, t^2NotOk, t^{24}Normotense, t^{24}NormalTemperature,λ},

γ: $\{[n_0, \lambda, n_1n_3n_5]$, $[n_1, t^8Hypotense, n_1]$, $[n_1, t^8Hypertense, n_1]$, $[n_1, Normotense, n_2]$, $[n_2, t^{24}Normotense, n_2]$,
$[n_2, Hypertense, n_1]$,
$[n_2, Hypotense, n_1]$, $[n_3, t^4Fever, n_3]$,
$[n_3, NormalTemperature, n_4]$,
$[n_4, t^{24}NormalTemperature, n_4]$,

[1]In order to simplify the example notation, it is assumed that each single clock has as performance environment its own state. Each symbol t^* is inherently associate to the clock that produced it and it was not considered that it affects on the other states derivations

$[n_4, Fever, n_3]$, $[n_5, t^2NotOk, n_5]$, $[n_5, t^{24}, n_6]$, $[n_5, Ok, n_7]$, $[n_6, t^{24}, n_7]$, $[n_2n_4n_7, \lambda, n_8]\}$,

δ^2: $\{[q_0, \lambda, q_{135}]$, $[q_{135}, t^8Hypotense, q_{135}]$, $[q_{135}, t^8Hypertense, q_{135}]$, $[q_{135}, t^4Fever, q_{135}]$, $[q_{135}, t^2NotOk, q_{135}]$, $[q_{135}, Normotense, q_{235}]$, $[q_{135}, NormalTemperature, q_{145}]$, $[q_{135}, t^{24}, q_{136}]$, $[q_{135}, Ok, q_{137}]$, $[q_{235}, t^{24}Normotense, q_{235}]$, $[q_{235}, t^4Fever, q_{235}]$, $[q_{235}, t^2NotOk, q_{235}]$, $[q_{235}, Hypertense, q_{135}]$, $[q_{235}, Hypotense, q_{135}]$, $[q_{235}, NormalTemperature, q_{245}]$, $[q_{235}, t^{24}, q_{236}]$, $[q_{235}, Ok, q_{237}]$, $[q_{145}, t^8Hypotense, q_{145}]$, $[q_{145}, t^8Hypertense, q_{145}]$, $[q_{145}, t^{24}NormalTemperature, q_{145}]$, $[q_{145}, t^2NotOk, q_{145}]$, $[q_{145}, Normotense, q_{245}]$, $[q_{145}, Fever, q_{135}]$, $[q_{145}, t^{24}, q_{146}]$, $[q_{145}, Ok, q_{147}]$, $[q_{136}, t^8Hypotense, q_{136}]$, $[q_{136}, t^8Hypertense, q_{136}]$, $[q_{136}, t^4Fever, q_{136}]$, $[q_{136}, Normotense, q_{236}]$, $[q_{136}, NormalTemperature, q_{146}]$, $[q_{136}, t^{24}, q_{137}]$, $[q_{137}, t^8Hypotense, q_{137}]$, $[q_{137}, t^8Hypertense, q_{137}]$, $[q_{137}, t^4Fever, q_{137}]$, $[q_{137}, Normotense, q_{237}]$, $[q_{137}, NormalTemperature, q_{147}]$, $[q_{245}, t24Normotense, q_{245}]$, $[q_{245}, t24NormalTemperature, q_{245}]$, $[q_{245}, t2NotOk, q_{245}]$, $[q_{245}, Hypotense, q_{135}]$, $[q_{245}, Hypertense, q_{135}]$, $[q_{245}, Fever, q_{235}]$, $[q_{245}, Ok, q_{247}]$, $[q_{245}, t^{24}, q_{246}]$, $[q_{236}, t24Normotense, q_{236}]$, $[q_{236}, t4Fever, q_{236}]$, $[q_{236}, Hypertense, q_{136}]$, $[q_{236}, Hypotense, q_{136}]$, $[q_{236}, NormalTemperature, q_{246}]$, $[q_{236}, t24, q_{237}]$, $[q_{237}, t24Normotense, q_{237}]$, $[q_{237}, t4Fever, q_{237}]$, $[q_{237}, Hypertense, q_{137}]$, $[q_{237}, Hypotense, q_{137}]$, $[q_{237}, NormalTemperature, q_{247}]$, $[q_{146}, t^8Hypotense, q_{146}]$, $[q_{146}, t^8Hypertense, q_{146}]$, $[q_{146}, t^{24}NormalTemperature, q_{146}]$, $[q_{146}, Normotense, q_{246}]$, $[q_{146}, Fever, q_{136}]$, $[q_{146}, t24, q_{147}]$, $[q_{147}, t^8Hypotense, q_{147}]$, $[q_{147}, t^8Hypertense, q_{147}]$, $[q_{147}, t^{24}NormalTemperature, q_{147}]$, $[q_{147}, Normotense, q_{247}]$, $[q_{147}, Fever, q_{137}]$, $[q_{246}, t24Normotense, q_{246}]$, $[q_{246}, t24NormalTemperature, q_{246}]$, $[q_{246}, Hypotense, q_{136}]$, $[q_{246}, Hypertense, q_{136}]$, $[q_{246}, Fever, q_{236}]$, $[q_{246}, t^{24}, q_{247}]$, $[q_{247}, \lambda, q_8]$ $\}$

$q_0 = \{q_0\}$,

$F = \{n_8\}$

The TPA main advantage over the previous models is time management. In addition, TPA, as PFA, is a good framework to represent guided steps over an Activity-Based Workflow Mining problem. Nevertheless, TPA and PFA, have not the capability to manage multithreading processes as Petri Nets can do, but this are not specifically needed on Activity-Based Workflow Mining.

Theorem 3.3 *The language class defined by the Timed Parallel Automaton is equivalent to the Regular Languages class.*

Proof 3.4 $AFN \subseteq TPA$*: It is obvious that a DFA is a limited TPA.*

$TPA \subseteq AFN$*: It exists an Non Deterministic Finite Automaton that accepts the language accepted by a TPA. Following the proof realized on [SP94] saying a PFA is regular. Based on this, it is easy to proof that a TPA has a PFA equivalent:*

A TPA T=$\{C_T, P_T, N_T, Q_T, T_T, \Phi_T, \Sigma_T, \gamma_T, q_{0T}, F_T\}$ *has a PFA equivalent P =*$\{N_P, Q_P, \Sigma_P, \gamma_P, \delta_P, q_{0P}, F_P\}$*:*

$N_P = N_T$

$Q_P = Q_T$,

$\Sigma_P = \Sigma_T \cup T_T$,

$\gamma_P = \gamma_T$,

$\delta_P = \delta_T$,

$q_{0P} = q_{0T}$,

$F_P = F_T$.

Theorem 3.5 *For each language accepted by a 1-safe Petri Net, exists a TPA that accept the same language.*

Proof 3.6 *In [SP94] it is proved that a PFA is equivalent to a 1-safe Petri Net. It is obvious that a PFA is a TPA without temporal information.*

In [vdABtHK03] [vdABtHK00] it is presented the requirements that must be fulfilled in a workflow language and engine to have a good expressive capability. Within this idea, in these papers, twenty patterns of workflow appear. They define different situations that workflow would have to solve. In conclusions, different commercial applications are analyzed in order to know how many workflow patterns fulfill. None of the commercial approaches analyzed fulfill all the workflow patterns.

Following, it is demonstrated how the defined automata is able to fulfill the Aalst patterns required expressivity. So,

[2] In order to make clearer the model the value λ is taken as an immediate step, in that way, although that in some other nodes might be other possible derivations, these would never be executed, reason why they have not been added

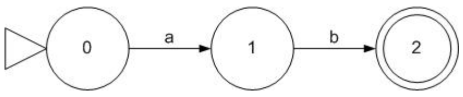

Figure 2. Sequence pattern

each one of those patterns will be enumerated in order to proof using examples this expressivity in formal and graphical way. The twenty defined patterns are presented in six different groups:

Basic Flow Patterns

The Basic Flow Patterns are those that deal with control processes elementary aspects. Those patterns appeared listed below:

Pattern 1: Sequence

The Sequence Pattern describes the simplest process in a control flow. This pattern express the process which an activity in a workflow process is enabled after the completion of another activity in the same process.

In the figure 2 is is shown graphically a Sequence Pattern example. A formal representation is presentes following:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0,1,2},
N = $\{n_0 : [0, c^\infty], n_1 : [1, c^\infty], n_2 : [2, c^\infty]\}$,
Q = $\{q_0 : n_0, q_1 : n_1, q_2 : n_2\}$,
T = $\emptyset$,
$\Phi = \{a, b\}$,
$\Sigma = \{a, b, \lambda\}$,
$\gamma : \{[n_0, a, n_1], [n_1, b, n_2]\}$,
$\delta : \{[q_0, a, q_1], [q_1, b, q_2]\}$,
$q_0 = \{q_0\}$,
$F = \{n_2\}$

As it is exposed, this formal representation is quite simple. The functions γ and δ are defined using the same arcs to the same nodes, it is to say, there are no differences between nodes and states vision. Those arcs define the automaton defined sequence.

Pattern 2: Parallel Split

The Parallel Split Pattern describe a process in which a point in the workflow process where a single control thread splits into multiple control threads which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.

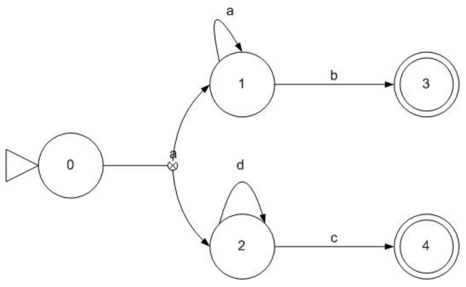

Figure 3. Parallel Split pattern

A Parallel Split pattern example appears in the figure 3 . In that image, the automaton changes from the initial node *0* with the *a* symbol, to the nodes *1* and *2* concurrently. A formal representation of the example can be observed below:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0,1,2,3,4},
N = $\{n_0 : [0, c^\infty]$, $n_1 : [1, c^\infty]$, $n_2 : [2, c^\infty]$, $n_3 : [3, c^\infty]$, $n_4 : [4, c^\infty]\}$,
Q = $\{q_0 : n_0$, $q_{12} : n_1n_2$, $q_{23} : n_2n_3$, $q_{14} : n_1n_4$, $q_{34} : n_3n_4\}$,
T = $\emptyset$,
$\Phi = \{a, b, c, d\}$,
$\Sigma = \{a, b, c, d, \lambda\}$,
γ: $\{[n_0, a, n_1n_2]$, $[n_1, a, n_1]$, $[n_2, d, n_2]$, $[n_1, b, n_3]$, $[n_2, c, n_4]$ $\}$,
δ: $\{[q_0, a, q_{12}]$, $[q_{12}, a, q_{12}]$, $[q_{12}, d, q_{12}]$, $[q_{12}, b, q_{23}]$, $[q_{12}, c, q_{14}]$, $[q_{23}, c, q_{34}]$, $[q_{14}, b, q_{34}]\}$,
$q_0 = \{q_0\}$,
$F = \{n_3, n_4\}$

This formal representation is a clear example of differences between nodes and states. While the nodes represent activities, the states are set of activities. The γ function constitute the node level transitions, as can be seen in the figure 3, nevertheless, the δ function denotes the transitions between states, counting all the transition possibilities between nodes, that can be executed concurrently.

Pattern 3: Synchronization

The Synchronization pattern is associated to process that have a point in the workflow process where multiple par-

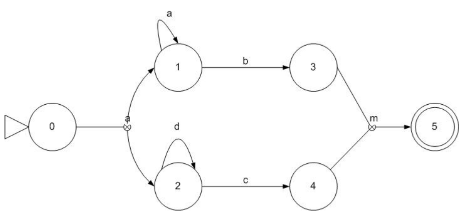

Figure 4. Synchronization pattern

allel subprocesses/activities converge into one single control thread, thus synchronizing multiple threads. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once. A Synchronization pattern example is illustrated in the figure 4. In the graph a Parallel Split process that divides the initial process is shown. The Synchronization process join both sequences in one, converging all in only one thread.A formal representation of the example is defined below:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^{\infty}\}$,
P = {0,1,2,3,4,5},
N = $\{n_0 : [0, c^{\infty}]$, $n_1 : [1, c^{\infty}]$, $n_2 : [2, c^{\infty}]$, $n_3 : [3, c^{\infty}]$, $n_4 : [4, c^{\infty}]$, $n_5 : [5, c^{\infty}]\}$,
Q = $\{q_0 : n_0$, $q_{12} : n_1 n_2$, $q_{23} : n_2 n_3$, $q_{14} : n_1 n_4$, $q_{34} : n_3 n_4$, $q_5 : n_5\}$,
T = $\emptyset$,
$\Phi = \{a, b, c, d, m\}$,
$\Sigma = \{a, b, c, d, m, \lambda\}$,
γ: $\{[n_0, a, n_1 n_2]$, $[n_1, a, n_1]$, $[n_2, d, n_2]$, $[n_1, b, n_3]$, $[n_2, c, n_4]$, $[n_3 n_4, m, n_5]$ $\}$,
δ: $\{[q_0, a, q_{12}]$, $[q_{12}, a, q_{12}]$, $[q_{12}, d, q_{12}]$, $[q_{12}, b, q_{23}]$, $[q_{12}, c, q_{14}]$, $[q_{23}, c, q_{34}]$, $[q_{14}, b, q_{34}]$, $[q_{34}, m, q_5]\}$,
$q_0 = \{q_0\}$,
$F = \{q_5\}$

In this formal representation is indicated how the Parallel Split and Synchronization patterns are defined into TPA. As in the γ function, in the δ function, the automaton change from the nodes *3* and *4*, where the concurrent activities sequences finalize, with the symbol *m* to the final node *5* only when the nodes *3* and *4* are reached.

Pattern 4: Exclusive Choice

This pattern describe a point in the workflow process where, based on a decision or workflow control data, one o several branches is chosen, it is to say, the Exclusive Choice pattern

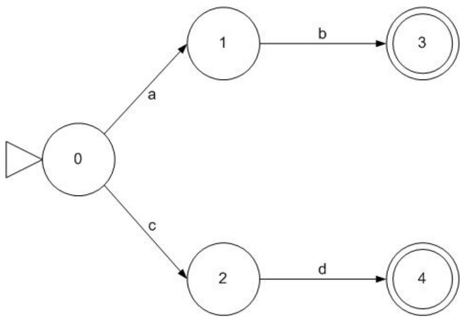

Figure 5. Exclusive Choice pattern

allows workflows to decide execute activities depending of the branches chosen.

A Exclusive Choice pattern is drawn in the figure 5.The step from the node *0* to the node *1* is produced when a *a* symbol is produced, on the other hand when a symbol *c* is produced at node *0*, the node *2* is reached. The formal representation of the example is described following:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^{\infty}\}$,
P = {0,1,2,3,4,5},
N = $\{n_0 : [0, c^{\infty}], n_1 : [1, c^{\infty}], n_2 : [2, c^{\infty}], n_3 : [3, c^{\infty}], n_4 : [4, c^{\infty}]\}$,
Q = $\{q_0 : n_0, q_1 : n_1, q_2 : n_2, q_3 : n_3, q_4 : n_4\}$,
T = $\emptyset$,
$\Phi = \{a, b, c, d\}$,
$\Sigma = \{a, b, c, d, \lambda\}$,
$\gamma : \{[n_0, a, n_1], [n_1, b, n_3], [n_0, c, n_2], [n_2, d, n_4]\}$,
$\delta : \{[q_0, a, q_1], [q_1, b, q_3], [q_0, c, q_2], [q_2, d, q_4]\}$,
$q_0 = \{q_0\}$,
$F = \{q_3, q_4\}$

As can be seen in the formal representation, it is very simple to represent this pattern using TPAs, adding an arc for each decision.

Pattern 5: Simple Merge

The Simple Merge pattern is associated to process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel.

A Simple Merge pattern example is defined on the figure6 In this case, a 2-branches Simple Merge pattern from nodes *1* and *2* to node *3*. A formal representation of the examples is presented following:

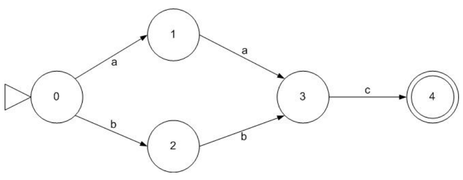

Figure 6. Simple Merge pattern

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0,1,2,3,4},
N = $\{n_0 : [0, c^\infty]$, $n_1 : [1, c^\infty]$, $n_2 : [2, c^\infty]$, $n_3 : [3, c^\infty]$, $n_4 : [4, c^\infty]\}$,
Q = $\{q_0 : n_0$, $q_1 : n_1$, $q_2 : n_2$, $q_3 : n_3$, $q_4 : n_4\}$,
T = $\emptyset$,
$\Phi = \{a, b, c\}$,
$\Sigma = \{a, b, c, \lambda\}$,
γ: $\{[n_0, a, n_1]$, $[n_0, b, n_2]$, $[n_1, a, n_3]$, $[n_2, b, n_3]$, $[n_3, c, n_4]$ },
δ: $\{[q_0, a, q_1]$, $[q_0, b, q_2]$, $[q_1, a, q_3]$, $[q_2, b, q_3]$, $[q_3, c, q_4]$ },
$q_0 = \{q_0\}$,
$F = \{n_4\}$

As it can be observed, it is very simple to represent this pattern using TPAs. It is only needed add arcs to the common node. Its important to emphasize the differences between this pattern and the Synchronization pattern. While the Synchronization pattern continuous with only one thread, the Simple Merge pattern, can deal with several threads executing on the same branch.

Advanced branching and synchronization patterns

Although the advanced branching and synchronization patterns are not part of the basic set flow patterns due to his complexity, this do not mean that its use was minority. This set of four patterns is usually used in the definition of real life processes. Its definition and TPA model example was presented in this section.

Pattern 6: Multi-choice

The Multi-choice pattern propose an improvement of the Exclusive Choice pattern. While the Exclusive Choice pattern allows to execute only one activities sequence at time, the Multi-choice pattern is able to execute an indeterminate

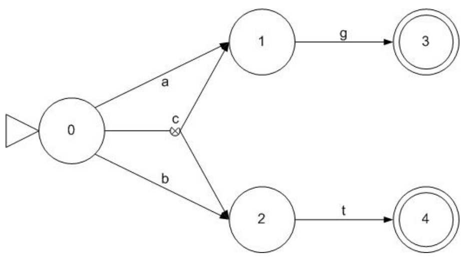

Figure 7. Multi-choice pattern

number of sequences at the same time, depending on the selection on execution time.

A Multi-choice example pattern is presented in the figure 7. The step from the node *0* to the nodes *1* and *2* is regulated by three arcs, the marked by the symbol *a* which aims to the node *1*, the marked by the symbol *b* which aims to the node *2*, and the marked by the symbol c, which aims to a *1* and *2* nodes parallel execution. Each arc marks a process execution possibility, defined by the used symbol. A formal representation of the example is presented below:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0,1,2,3,4},
N = $\{n_0 : [0, c^\infty]$, $n_1 : [1, c^\infty]$, $n_2 : [2, c^\infty]$, $n_3 : [3, c^\infty]$, $n_4 : [4, c^\infty]\}$,
Q = $\{q_0 : n_0, q_1 : n_1, q_2 : n_2, q_3 : n_3, q_4 : n_4$, $q_{12} : n_1n_2$, $q_{34} : n_3n_4$, $q_{14} : n_1n_4$, $q_{23} : n_2n_3\}$,
T = $\emptyset$,
$\Phi = \{a, b, c, g, t\}$,
$\Sigma = \{a, b, c, g, t, \lambda\}$,
γ: $\{[n_0, a, n_1]$, $[n_0, b, n_2]$, $[n_0, c, n_1n_2]$, $[n_1, g, n_3]$, $[n_2, t, n_4]\}$,
δ : $\{[q_0, a, q_1]$, $[q_0, b, q_2]$, $[q_0, c, q_{12}]$, $[q_1, g, q_3]$, $[q_2, t, q_4]$, $[q_{12}, t, q_{14}]$, $[q_{12}, g, q_{23}]$, $[q_{14}, g, q_{34}]$, $[q_{23}, t, q_{34}]$ },
$q_0 = \{q_0\}$,
$F = \{n_3, n_4\}$

This formal representation uses different states for a different workflow execution ways, it is to say, while at the node point of view the use of one transition or another is indifferent, at the state point of view the states set that deal with the transition is different.

Pattern 7: Synchronizing Merge

The Synchronizing Merge pattern works in analog way to the Multi-choice pattern, but where multiple paths converge

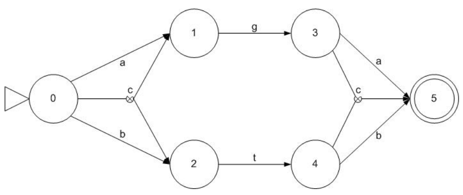

Figure 8. Synchronizing Merge patterns

into one single thread. When in a process a Multi-choice pattern is executed, it is probable that some time after the potentially selected to be executed sequenced activities set, converge in one point. In this case, the main problem is to know how the sequences must continue, it is to say, what sequences must be merged. The Synchronicing Merge pattern defines the several execution threads merge processes, synchronizing it, depending on its execution.

In the figure 8 a Synchronizing Merge pattern example is shown. In the example, it is possible to see how is divided the initial sequence in two activity sequences, according to the generated symbols, and how both sequences are joined as corresponding in its execution. A formal representation of the problem is presented following:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0,1,2,3,4,5},
N = $\{n_0 : [0, c^\infty], n_1 : [1, c^\infty], n_2 : [2, c^\infty], n_3 : [3, c^\infty], n_4 : [4, c^\infty], n_5 : [5, c^\infty]\}$,
Q = $\{q_0 : n_0, q_1 : n_1, q_2 : n_2, q_3 : n_3, q_4 : n_4, q_5 : n_5, q_{12} : n_1n_2, q_{34} : n_3n_4, q_{14} : n_1n_4, q_{23} : n_2n_3\}$,
T = $\emptyset$,
$\Phi = \{a, b, c, g, t\}$,
$\Sigma = \{a, b, c, g, t, \lambda\}$,
γ: $\{[n_0, a, n_1], [n_0, b, n_2], [n_0, c, n_1n_2], [n_1, g, n_3], [n_2, t, n_4], [n_3, a, n_5], [n_3n_4, c, n_5], [n_4, b, n_5]\}$,
δ : $\{[q_0, a, q_1], [q_0, b, q_2], [q_0, c, q_{12}], [q_1, g, q_3], [q_2, t, q_4], [q_{12}, t, q_{14}], [q_{12}, g, q_{23}], [q_{14}, g, q_{34}], [q_{23}, t, q_{34}], [q_3, a, q_5], [q_4, b, q_5], [q_{34}, c, q_5]\}$,
$q_0 = \{q_0\}$,
$F = \{n_5\}$

As it was shown before, the Multi-choice pattern suppose the possible ways split in nodes sets, in order to merge them, it is needed to add arcs from the different nodes sets to the final activities sequence.

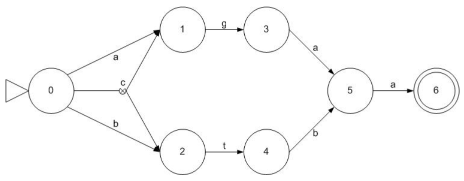

Figure 9. Multi-merge pattern

Pattern 8: Multi-merge

The Multi-merge pattern is a functional alternative to the Synchronizing Merge pattern. While the Synchronizing Merge assumes that the activities sequences that was selected at the workflow beginning must finish with synchrony, the Multi-merge pattern allows the sequences merging without synchronization.

A Multi-merge example is shown in the figure9. In the example, the initial sequence is split, depending the symbol, up to two sequences, and after, are merged separately without synchronization arcs. A formal representation is described following:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0,1,2,3,4,5,6},
N = $\{n_0 : [0, c^\infty], n_1 : [1, c^\infty], n_2 : [2, c^\infty], n_3 : [3, c^\infty], n_4 : [4, c^\infty], n_5 : [5, c^\infty], n_6 : [6, c^\infty]\}$,
Q = $\{q_0 : n_0, q_1 : n_1, q_2 : n_2, q_3 : n_3, q_4 : n_4, q_5 : n_5, q_6 : n_6\ q_{12} : n_1n_2, q_{34} : n_3n_4, q_{14} : n_1n_4, q_{23} : n_2n_3, q_{15} : n_1n_5, q_{35} : n_3n_5, q_{25} : n_2n_5, q_{45} : n_4n_5, q_{16} : n_1n_6, q_{36} : n_3n_6, q_{26} : n_2n_6, q_{46} : n_4n_6, q_{56} : n_5n_6, q_{55} : n_5n_5, q_{66} : n_6n_6\ \}$,
T = $\emptyset$,
$\Phi = \{a, b, c, g, t\}$,
$\Sigma = \{a, b, c, g, t, \lambda\}$,
γ: $\{[n_0, a, n_1], [n_0, b, n_2], [n_0, c, n_1n_2], [n_1, g, n_3], [n_2, t, n_4], [n_3, a, n_5], [n_4, b, n_5], [n_5, a, n_6]\}$,
δ : $\{[q_0, a, q_1], [q_0, b, q_2], [q_0, c, q_{12}], [q_1, g, q_3], [q_2, t, q_4], [q_{12}, t, q_{14}], [q_{12}, g, q_{23}], [q_{14}, g, q_{34}], [q_{25}, t, q_{45}], [q_{15}, g, q_{35}], [q_{35}, a, q_{56}], [q_{45}, b, q_{55}], [q_{56}, a, q_6], [q_{45}, a, q_{46}], [q_{15}, g, q_{35}], [q_{46}, b, q_5], [q_{26}, t, q_4], [q_{16}, g, q_3], [q_{36}, a, q_5], [q_{55}, a, q_{66}]\ \}$,
$q_0 = \{q_0\}$,
$F = \{n_6\}$

The Multi-merge pattern utilization, must keep in mind all the process execution possibilities, taking into account

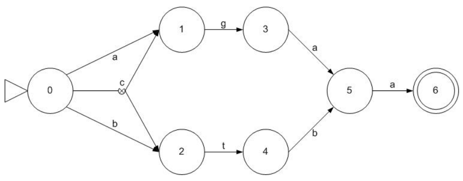

Figure 10. Discriminator Pattern

the activities executes after the Multi-merge pattern, since, they can be executed more than once in a out of phase way

Pattern 9: Discriminator

When a Multi-merge pattern that executes more than one activities sequences at the same time is applied, the activities executed after the pattern are executed more than once is not always desired. The most obvious alternative to this problem is the Synchronizing Merge pattern, nevertheless, it is not always desired too, to wait for all the activities end to resume the process. So, the Discriminator pattern arise to solve it. This pattern, allows continue the first sequence that reach it, finishing the rest of sequences that reach it. In this way the activity execution duplicity problems are solved.

A Discriminator pattern example is shown in figure 10. Although the graphic can look like similar to the Multi-merge pattern, there are differences into the formal definition:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0,1,2,3,4,5,6},
N = $\{n_0 : [0, c^\infty]$, $n_1 : [1, c^\infty]$, $n_2 : [2, c^\infty]$, $n_3 : [3, c^\infty]$, $n_4 : [4, c^\infty]$, $n_5 : [5, c^\infty]$, $n_6 : [6, c^\infty]\}$,
Q = $\{q_0 : n_0$, $q_1 : n_1$, $q_2 : n_2$, $q_3 : n_3$, $q_4 : n_4$, $q_5 : n_5$, $q_6 : n_6$ $q_{12} : n_1n_2$, $q_{34} : n_3n_4$, $q_{14} : n_1n_4$, $q_{23} : n_2n_3$, $q_{15} : n_1n_5$, $q_{35} : n_3n_5$, $q_{25} : n_2n_5$, $q_{45} : n_4n_5$, $q_{16} : n_1n_6$, $q_{36} : n_3n_6$, $q_{26} : n_2n_6$, $q_{46} : n_4n_6\}$,
T = $\emptyset$,
$\Phi = \{a, b, c, g, t\}$,
$\Sigma = \{a, b, c, g, t, \lambda\}$,
γ: $\{[n_0, a, n_1]$, $[n_0, b, n_2]$, $[n_0, c, n_1n_2]$, $[n_1, g, n_3]$, $[n_2, t, n_4]$, $[n_3, a, n_5]$, $[n_4, b, n_5]$, $[n_5, a, n_6]\}$,
δ : $\{[q_0, a, q_1]$, $[q_0, b, q_2]$, $[q_0, c, q_{12}]$, $[q_1, g, q_3]$, $[q_2, t, q_4]$, $[q_{12}, t, q_{14}]$, $[q_{12}, g, q_{23}]$, $[q_{14}, g, q_{34}]$, $[q_{23}, t, q_{34}]$, $[q_{23}, a, q_{25}]$, $[q_{14}, b, q_{15}]$, $[q_{34}, b, q_{35}]$, $[q_{34}, a, q_{45}]$, $[q_{15}, g, q_{35}]$, $[q_{25}, t, q_{45}]$, $[q_{35}, a, q_6]$, $[q_{45}, a, q_6]$, $[q_{25}, a, q_{26}]$, $[q_{15}, a, q_{16}]$, $[q_{16}, g, q_{36}]$, $[q_{26}, t, q_{46}]\}$,
$q_0 = \{q_0\}$,
$F = \{n_6\}$

In this formal description, while in the Multi-merge pattern is allowed the all ways execution, the Discriminator pattern prevent the activities sequences resume after the pattern is reached.

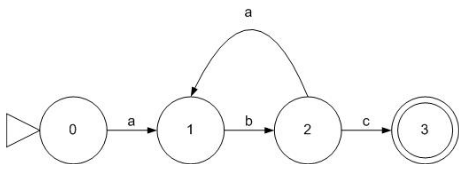

Figure 11. Arbitrary Cycles pattern

Structural patterns

Some workflow management systems impose different restrictions on their workflow models, as an example,prevent arbitrary cycles on workflow definitions. Those restrictions are not always natural from a modeling point of view, and tend to restrict the specification freedom to the workflow designers. In this section, are presented two patterns that represent typical workflow management systems structural restrictions, which, on the other hand, usually are interesting to the workflows definition.

Pattern 10: Arbitrary Cycles

An Arbitrary Cycle is a point in a workflow process where one or more activities can be executed repeatedly.

In figure 11 it is shown an Arbitrary Cycle pattern example. A TPA do not have any restriction to the arcs utilization in order to execute processes,in this way, it is possible to execute complex cycles. The example formal description is presented next.

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0, 1, 2, 3},
N = $\{n_0 : [0, c^\infty]$, $n_1 : [1, c^\infty]$, $n_2 : [2, c^\infty]$, $n_3 : [3, c^\infty]\}$,
Q = $\{q_0 : n_0$, $q_1 : n_1$, $q_2 : n_2$, $q_3 : n_3\}$,
T = $\emptyset$,
$\Phi = \{a, b, c\}$,
$\Sigma = \{a, b, c, \lambda\}$,
γ: $\{[n_0, a, n_1]$, $[n_1, b, n_2]$, $[n_2, a, n_1]$, $[n_2, c, n_3]\}$,

δ: $\{[q_0, a, q_1],\quad [q_1, b, q_2],\quad [q_2, a, q_1],$
$[q_2, c, q_3]\}$,
$q_0 = \{q_0\}$,
$F = \{n_3\}$

In that example, to define the cycle, it is only needed to create an arc from the final node to the initial node.

Pattern 11: Implicit Termination

In some workflow engines a process only finish when the process reach a final state. This is not always desirable at implementation level, since, at times, not normal finished processes must be forced to finish in,even though the state reached is not final. The Implicit Termination pattern allow to terminate a process, not only when a final stated was reached, but also when there are no active activities, in the workflow and no other activity can be made active, and at the same time the workflow is not in deadlock.

The Implicit Termination problem is an engine implementation level problem, and for this, it not affects to workflow design level, in this way there are no a formal representation associated in a TPA.

Patterns involving multiple instances

From a theoretical point of view the multiple instance patterns corresponds to multiple threads of execution referring to a shared definition. Nevertheless, from a practical point of view, its implementation is no so easy due, over all, to a design restrictions.

When patterns that involve multiple instances, it was considered two kind of abilities: abilities to launch multiples activities or subprocesses instances, or abilities to synchronizing those activities. All the patterns described in this section fulfill the first ability, however, the main differences between the patterns presented in this section are depending of the activities or subprocesses synchronization capability.

In this section patterns involving multiples instances are defined. Nevertheless, although this problem in almost all cases can be solved using the representation power of the TPA, usually is better to deal with this problem at engine implementation level, because, in practice, is not a formal framework representation level problem. In addition, is easier to deal with this problem at engine implementation level. In fact, if this separation is done and this problem is dealed at execution time, all the workflows will be scalable easily without the workflow design change necessity.

Pattern 12: Multiple Instances Without Synchronization

The Multiple Instances Without Synchronization pattern defines a workflow within the single case context, multiple instances of an activity can be created. Each of these control threads are independent of other threads. Moreover, there is no need to synchronize these threads.

In this pattern, it is no need to synchronize the instances, for this, there are no real difference to execute one workflow with all the instances or execute all the workflow instances in different threads. In this case, a TPA interpretation engine, can execute TPAs in different threads.

Pattern 13: Multiple Instances with a Priori Design Time Knowledge

The Multiple Instances with a Priori Design Time Knowledge defines a workflow that for one process instance an activity is enabled multiple times. The given activity instances number is known at design time. Once all Activities are completed, some other activity needs to be started.

In this case, the process synchrony is necessary to deal with the problem. At formal representation level is possible to deal with this, because the given activity instances number is known at design time (and this means that is finite), for this is possible to create a TPA that takes into account all the execution processes possibilities. Nevertheless, in practice, if we deal with this problem at interpretation engine level, the system will be more easy and scalable. The problem can be solved, using the interpretation engine, and two patterns that will be well defined after: the Deferred Choice pattern (that allows a third part to do a choose) and the Milestone pattern (that allows a workflow to wait up to some specific state is reached). In this case, the workflow designer can incorporate at the synchronization point a Milestone pattern with a Deferred Choice pattern that is enabled by the engine allowing the workflow instances to be resumed.

Pattern 14: Multiple Instances With a Priori Runtime Knowledge

The Multiple Instances With a Priori Runtime Knowledge defines a workflow which for one case an activity is enabled multiple times. The given activity instances number varies and may depend on case characteristics or resources availability, but is known during runtime, before the instances of that activity have to be created. Once all instances are completed some other activity needs to be started.

This pattern can be solved using the engine in the same way an shown in the pattern 13.

Pattern 15: Multiple Instances Without a Priori Runtime Knowledge

The Multiple Instances Without a Priori Runtime Knowledge defines a workflow that for one case an activity in enabled multiple times. The given activity instances number

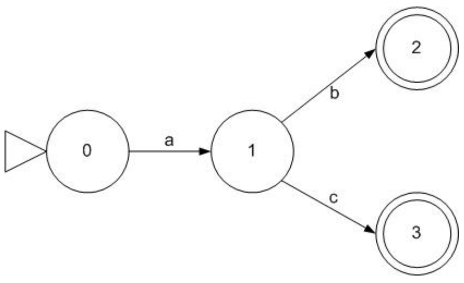

Figure 12. Deferred Choice pattern

is not known at design time nor at execution time. Once all instances are completed some other activity needs to be started.

This pattern can be solved using the engine in the same way than the pattern 13.

State-based patterns

In real workflows, most workflows instances are in a state awaiting processing rather than being processes. The state-based patterns are three patterns in which the processes can be indefinitely waiting in a state, depending to internal or external factors.

In a TPA the waiting state concept is produced just at final of activity execution. At this moment, the process is awaiting for a symbol to pass to other state.

The state-based patterns are listed below:

Pattern 16: Deferred Choice

The Deferred Choice pattern defines the process where one of several branches is chosen where the choice is not made explicitly. This choice can be realized by external or internal processes. It is important to note that the choice is delayed until the processing in one alternative branches is actually started.

In figure 12 a Deferred Choice pattern example is shown. In this graph it is possible to reach from node *1* to node *2* with a symbol *b* or to node *3* with a symbol *c*. As can be seen this pattern is equivalent to the Exclusive Choice pattern, but, in this case, the next branch is selected by a external process. A formal representation can be presented following:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0, 1, 2, 3},
N = $\{n_0 : [0, c^\infty]$, $n_1 : [1, c^\infty]$, $n_2 : [2, c^\infty]$, $n_3 : [3, c^\infty]\}$,
Q = $\{q_0 : n_0$, $q_1 : n_1$, $q_2 : n_2$, $q_3 : n_3\}$,
T = $\emptyset$,
$\Phi = \{a, b, c\}$,
$\Sigma = \{a, b, c, \lambda\}$,
γ: $\{[n_0, a, n_1]$, $[n_1, b, n_2]$, $[n_1, c, n_3]\}$,
δ: $\{[q_0, a, q_1]$, $[q_1, b, q_2]$, $[q_1, c, q_3]\}$,
$q_0 = \{q_0\}$,
$F = \{n_2, n_3\}$

At the formal point of view, there are not differences between this pattern and the exclusive choice pattern. In this case, the pattern can stand waiting indefinitely in the node *1* up to one external process or user, capable to interactuate with the process, generate the needed symbol to resume the workflow.

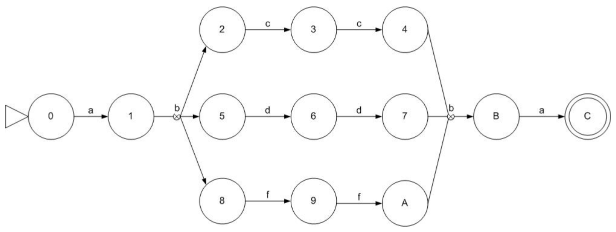

Figure 13. Interleaved Parallel Routing pattern

Pattern 17: Interleaved Parallel Routing

The Interleaved Parallel Routing pattern defines a process in which a set of activities is executed in an arbitrary order, where, each activity in the set is executed, the order is decided at runtime, and not two activities are active for the same workflow instance at the same time.

A Interleaved Parallel Routing pattern example is presentes in the figure 13. In this example exist three activities sequences that corresponds to nodes *2-3-4*, *5-6-7*, and *8-9-A* , that must to be executed in an arbitrary order, but not at the same time. The sequences initial states *2, 5 and 8* are the waiting states where the activities wait to be executed. On the other hand the sequences final states *4, 7, and A* are final waiting states of each of one of the activities sequences, where once executed, this sequences wait to the other sequences finalize. A formal representation is defined below:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$
C = $\{c^\infty\}$,
P = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C},
N = $\{n_0 : [0, c^\infty]$, $n_1 : [1, c^\infty]$, $n_2 : [2, c^\infty]$, $n_3 : [3, c^\infty]$, $n_4 : [4, c^\infty]$, $n_5 : [5, c^\infty]$, n_6 :

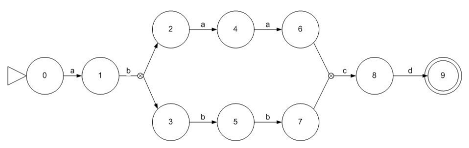

Figure 14. Milestone pattern

$[6, c^{\infty}]$, $n_7 : [7, c^{\infty}]$, $n_8 : [8, c^{\infty}]$, $n_9 : [9, c^{\infty}]$, $n_A : [A, c^{\infty}]$, $n_B : [B, c^{\infty}]$, $n_C : [C, c^{\infty}]\}$,

Q = $\{q_0 : n_0$, $q_1 : n_1$, $q_{258} : n_2n_5n_8$, $q_{259} : n_2n_5n_9$, $q_{25A} : n_2n_5n_A$, $q_{268} : n_2n_6n_8$, $q_{26A} : n_2n_6n_A$, $q_{278} : n_2n_7n_8$, $q_{279} : n_2n_7n_9$, $q_{27A} : n_2n_7n_A$, $q_{358} : n_3n_5n_8$, $q_{35A} : n_3n_5n_A$, $q_{378} : n_3n_7n_8$, $q_{37A} : n_3n_7n_A$, $q_{459} : n_4n_5n_9$, $q_{45A} : n_4n_5n_A$, $q_{468} : n_4n_6n_8$, $q_{46A} : n_4n_6n_A$, $q_{478} : n_4n_7n_8$, $q_{479} : n_4n_7n_9$, $q_{47A} : n_4n_7n_A$, $q_B : n_B$, $q_C : n_C\}$,

T = $\emptyset$,

$\Phi = \{a, b, c, d, f\}$,

$\Sigma = \{a, b, c, d, f, \lambda\}$,

γ: $\{[n_0, a, n_1]$, $[n_1, b, n_2n_5n_8]$, $[n_2, c, n_3]$, $[n_3, c, n_4]$, $[n_5, d, n_6]$, $[n_6, d, n_7]$, $[n_8, f, n_9]$, $[n_9, f, n_A]$, $[n_4n_7n_A, b, n_B]$, $[n_B, a, n_C]\}$,

δ: $\{[q_0, a, q_1]$, $[q_1, b, q_{258}]$, $[q_{258}, c, q_{358}]$, $[q_{258}, d, q_{268}]$, $[q_{258}, f, q_{259}]$, $[q_{358}, c, q_{458}]$, $[q_{268}, d, q_{278}]$, $[q_{259}, f, q_{25A}]$, $[q_{458}, d, q_{468}]$, $[q_{458}, f, q_{459}]$, $[q_{468}, d, q_{478}]$, $[q_{459}, f, q_{45A}]$, $[q_{378}, c, q_{478}]$, $[q_{35A}, c, q_{45A}]$, $[q_{278}, c, q_{378}]$, $[q_{278}, f, q_{279}]$, $[q_{25A}, c, q_{35A}]$, $[q_{25A}, d, q_{26A}]$, $[q_{478}, f, q_{479}]$, $[q_{46A}, d, q_{47A}]$, $[q_{45A}, d, q_{46A}]$, $[q_{279}, f, q_{27A}]$, $[q_{26A}, d, q_{27A}]$, $[q_{479}, f, q_{47A}]$, $[q_{37A}, c, q_{47A}]$, $[q_{27A}, c, q_{37A}]$, $[q_{47A}, b, q_B]$, $[q_B, a, q_C]$ },

$q_0 = \{q_0\}$,

$F = \{n_C\}$

The δ states function only have the possible transtitions avoiding the prohibited states,that are those in which there are sly activities.

Pattern 18: Milestone

The Milestone pattern is based on the idea in which the enabling of an activity depends on the case being in a specified state, it is to say, that an activity is only enabled if a certain milestone has been reached.

In figure 14 a Milestone pattern example is shown. In that example, the node *4* is taken as a Milestone for the sequence *3-5-7* at the node *5* and the node *5* is a Milestone for the sequence *2-4-6* at the node *4*. A formal specification is presented below:

A =$\{C, P, N, Q, T, \Phi, \Sigma, \gamma, q_0, F\}$

C = $\{c^{\infty}\}$,

P = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, H},

N = $\{n_0 : [0, c^{\infty}]$, $n_1 : [1, c^{\infty}]$, $n_2 : [2, c^{\infty}]$, $n_3 : [3, c^{\infty}]$, $n_4 : [4, c^{\infty}]$, $n_5 : [5, c^{\infty}]$, $n_6 : [6, c^{\infty}]$, $n_7 : [7, c^{\infty}]$, $n_8 : [8, c^{\infty}]$, $n_9 : [9, c^{\infty}]$, $n_H : [H, c^{\infty}]\}$,

Q = $\{q_0 : n_0$, $q_1 : n_1$, $q_{23} : n_2n_3$, $q_{34} : n_3n_4$, $q_{25} : n_2n_5$, $q_{45} : n_4n_5$, $q_{56} : n_5n_6$, $q_{47} : n_4n_7$, $q_{67} : n_6n_7$, $q_8 : n_8$, $q_9 : n_9$ },

T = $\emptyset$,

$\Phi = \{a, b, c, d\}$,

$\Sigma = \{a, b, c, d, \lambda\}$,

γ: $\{[n_0, a, n_1]$, $[n_1, b, n_2n_3]$, $[n_2, a, n_4]$, $[n_4, a, n_6]$, $[n_3, b, n_5]$, $[n_5, b, n_7]$, $[n_6n_7, c, n_8]$, $[n_8, d, n_9]\}$,

δ: $\{[q_0, a, q_1]$, $[q_1, b, q_{23}]$, $[q_{23}, a, q_{34}]$, $[q_{23}, b, q_{25}]$, $[q_{34}, b, q_{45}]$, $[q_{25}, a, q_{45}]$, $[q_{45}, a, q_{56}]$, $[q_{45}, b, q_{47}]$, $[q_{47}, a, q_{67}]$, $[q_{56}, b, q_{67}]$, $[q_{67}, c, q_8]$, $[q_8, d, q_9]$ },

$q_0 = \{q_0\}$,

$F = \{n_9\}$

In the formal specification δ function it is possible to see the state *45* as a Milestone. In that formal specification, only are valid the arcs that define the pass through the nodes *4* and *5*. In an analog way, it is possible to use only a Milestone, and one of the sequences have not restrictions to continue.

Cancelation patterns

The cancelation patterns are referred to the activities or instances cancel possibility at execution time.

Those patterns are totally engine dependent, and for this there are any difference at formal level. Nevertheless, the cancelation patterns are listed below:

Pattern 19: Cancel Activity

The Cancel Activity pattern allows to disable an activity that was enabled at execution time

Pattern 20: Cancel Case

The Cancel Case pattern allows to cancel the execution of an instance that was executed in the interpretation engine.

4 Conclusion

In this paper a new paradigm called, Activity-Based Workflow Mining has been presented. This paragigm is able to model more complex systems than the classical Event-Based Workflow Mining based systems, and allow to model systems that are not contemplated in other disciplines.

On the other hand, a formal framework called, TPA, has been presented in this paper. This framework turns out interesting for designing Activities-Based Workflow Based systems, since they are better conformed than classical systems, plus its complexity is limited to the Regular Languages. In addition, this formal framework is according to the 20 Aalst workflow patterns.

The framework defined in this paper is being used in several projects in order to create a workflow interpretation model.

1. An Ambient Intelligence simulator is being developed to create activity-based workflow mining over AmI environments

2. A TPA interpretation model is being developed to be included into a teleasistence platform in collaboration with the regional government of Valencia in order to help Hospital la Fe, Home Hospitalization Unit, to implement guided medical care plans

The next step is building learning algorithms that could take profit of systems TPA Logs that are under developing state in order to build new workflows automatically using Activity-Based Workflow Mining.

References

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. Technical report, Computer Science Department Stanford University, 1994.

[AM04] Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. Technical report, University of Pensylvania, 2004.

[Coa99] Workflow Management Coalition. Terminology and glosary, 1999.

[CW98] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[D'A] Pedro R. D'Argenio. Regular processes and timed automata.

[GHS95] Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[GMA95] B. Grahlmann, M. Moeller, and U. Anhalt. A new interface for the pep tool – parallel finite automata –. In *2. Workshop Algorithmen und Werkzeuge für Petrinetze*, 22, pages 21–26, 1995.

[Gra98] Bernd Grahlmann. Combining finite automata, parallel programs and sdl using petri nets. In Bernhard Steffen, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *Lecture Notes in Computer Science*, pages 102–117. Springer-Verlag, 1998.

[JEM99] Petr Jancar, Javier Esparza, and Faron Moller. Petri nets and regular processes. *Journal of Computer and System Sciences*, 59(3):476–503, 1999.

[KtHB00] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In *Conference on Advanced Information Systems Engineering*, pages 431–445, 2000.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[MY01] H.G. Mendelbaum and R.B. Yehezkael. Using parallel automaton as a single notation to specify, design and control small computer based systems. In *IEEE 2001 - ECBS*, 2001.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Technical report, Cambridge, MA, USA, 1974.

[SP94] P. David Stotts and William Pugh. Parallel finite automata for modeling concurrent software systems. *The Journal of Systems and Software*, 27(1):27–??, October 1994.

[SRGH05] SE Straus, WS Richardson, Paul Glasziou, and RB Haynes. *Evidence-based Medicine:How to Practice and Teach EBM*. Churchill Livingstone, 3 edition, 2005.

[SVD01] Jan Springintveld, Frits Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1–2):225–257, 2001.

[vdA96] W. M. P. van der Aalst. Three good reasons for using a petri-net-based workflow management system. In Navathe, S. and Wakayama, T., editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96), Cambridge, Massachusetts, Nov. 14-15, 1996*, pages 179–201, 1996.

[vdABtHK00] Wil M. P. van der Aalst, Alistair P. Barros, Arthur H. M. ter Hofstede, and Bartek Kiepuszewski. Advanced workflow patterns. In *Conference on Cooperative Information Systems*, pages 18–29, 2000.

[vdABtHK03] Wil M. P. van der Aalst, Alistair P. Barros, Arthur H. M. ter Hofstede, and Bartek Kiepuszewski. Workflow patterns. page 70, 2003.

[vdAH02] W. van der Aalst and A. Hofstede. Yawl: Yet another workflow language, 2002.

[vdAvDH$^+$03] Wil M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and aproaches. pages 237–267, 2003.

[vdAWM03] W. van der Aalst, A. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs, 2003.

[WvdA] T. Weijters and W. van der Aalst. Rediscovering workflow models from event-based data.