2011 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises

Petri net modeling and verification of transactional workflows

Kais Klai
Information Department
LIPN, CNRS UMR 7030,
Université Paris 13
kais.klai@lipn.univ-paris13.fr

Walid Gaaloul
Information Department
TELECOM SudParis
UMR 5157 CNRS Samovar, France
walid.gaaloul@it-sudparis.eu

***Abstract*—The increasing use of Workflow Management Systems (WfMS) in companies expresses their undeniable importance to improve the efficiency of their processes and their execution costs. However, with the technological improvements and the continuous increasing market pressures and requirements, collaborative information systems are becoming more and more complex, involving numerous interacting business objects. Consequently, in spite of their obvious potential, WfMS show some limitations to ensure a correct and reliable execution. Therefore, there is a growing interest for verification techniques which help to provide reliable transactional workflow behavior and thereafter prevent workflow execution failures. In this paper, we propose a Petri net driven approach to validate workflow transactional behavior to subsequently improve and correct related recovery mechanisms. The transactional behavior verification is done at design time to validate the transactional behavior consistency and help designers to provide correct recovery mechanisms. By using Petri nets and Temporal Logic formalisms to specify and check the transactional behavior consistency, the approach we present in this paper provides a logical foundation to ensure workflow execution reliability.**

I. Introduction

The increasing use of Workflow Management Systems (WfMS) in companies expresses their undeniable importance to improve the efficiency of their processes and their execution costs. However, with the technological improvements and the continuous increasing market pressures and requirements, collaborative information systems are becoming more and more complex, involving numerous interacting business objects. Consequently, in spite of their obvious potential, WfMS show some limitations to ensure a correct and reliable execution. Due to the complex design process, it is impossible to easily foresee and initially realize all necessary parameters for a "perfect" design.

WfMS are expected to recognize and handle errors to support reliable and consistent executions of workflows. Since business processes contain activities that access shared and persistent data resources, they have to be subject to transactional semantics [1], [2], and must span multiple transaction models and protocols native to the underlying legacy systems upon which the workflows are dependent. As [3] pointed out, the introduction of some kind of transactions in WfMS is unavoidable to guarantee reliable and consistent workflow executions. Transactional workflows propose to combine workflow flexibility and transactional reliability in order to deal with these issues. Our main contribution in this paper is a new Petri net-based approach that helps designers for the modeling and the verification of transactional workflows. The input of our approach is the control flow of the model completed by the transactional properties and dependencies that have to be satisfied by the model. For each property and each dependency type a Petri net pattern is proposed completed by Linear Time Logic (LTL) formula to be checked on the Petri net representation. Thus, the output will be either a positive answer i.e. the model is correct and take in account all the expected recovery mechanisms, or negative; therefore a counter example (an execution that unsatisfy one or more expected behaviors) is supplied and the designer has to correct his model in consequence.

This paper is organized as follows: In Section II, a motivating example, on which our approach is applied, is introduced. In Section III some basic notions on Petri nets, transactional workflows and LTL verification are presented. Section IV is the core of the paper. The formal specification of transactional workflows is discussed. Our approach is then applied on the case study example in Section IV-E. Related works are discussed in Section V and, finally, Section VI concludes the paper and gives some future works.

II. Motivating Example

We consider a car rental scenario where one party acts as a broker based on the customer's earlier choice in the Customer Requirements Specification (CRS) activity. The Customer Identity Check (CIC) activity checks the customer ID while the Car Checking Availability (CCA) activity and the Parking Localisation (PL) provides information about available cars and the respective car rental companies supplier. Afterwards, the customer makes his choice and agrees on rental terms in the Car Agreement (CA) activity. Then, the customer is requested to pay either by Credit Card (CC), by CHeck (CH), or by caSH (SH). Finally, the bill is sent to the customer by the Send Bill (SB) activity (The WF-net corresponding to our running example control flow is shown in Figure 1).

In our example, four activities : CCA, CC, CA and SH can fail. Different recovery mechanisms **have to be correctly**

specified by the designers for these activities to recover a consistent state. For CCA failures (the workflow instance does not find any car propositions), the designers propose to cancel the CIC execution and to execute a backward recovery by restarting the instance execution to (re-)enter the client requirements workflow instance. After CA failure, the designers aim at implementing a backward recovery by restarting the car rental discovery process (CIC, CCA and PL) after executing a compensation activity (DBC), which compensates already executed activities [4]. To ensure the car rental payment in case of SH failure, the designers propose to specify the payment by credit card as an alternative. The designers can choose also a backward recovery from CA. As recovery mechanism for CC failure, the designers propose to add to this activity the capability to be (re-)executed until success in case of failure.

As illustrated through our motivating example, the transactional properties and dependencies of our model are as important as the control flow, and must be part of the modelling process to ensure the expected correct behavior. If several approaches dealing with modeling and verification of control workflow exists in the literature (e.g. [5], [6], [7], [8], ...), the latter is relatively poor in respect of transactional workflow design and verification. This is the core of our contribution in this paper. We propose to help the designers to formally specify the workflow using Petri nets. Also, we express the properties to be satisfied by the model for each kind of transactional dependency. These properties are expressed with Linear Time Logic (LTL) and can be checked with any of the existing model checker tools. This would allow us to verify and then detect potential transactional design gaps and correct them to provide a correct and reliable model. Indeed, such incorrect specifications of the transactional behavior can result in unpredictable behavior, which, in turn, can lead to unavailability of resources, information integrity problems and global workflow execution failure.

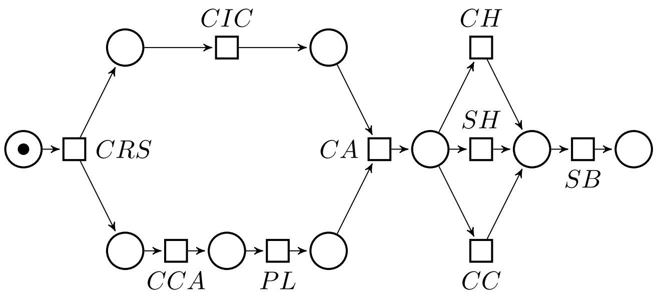

Figure 1. Control flow

III. Preliminaries

We give in this section, an informal description of our generic workflow transactional model followed by some preliminaries on Petri nets.

A. Transactional Workflow Model

Transactional workflow models have been introduced in [9] to clearly recognize the relevance of transactions in the context of workflows. In the following, we present a synthetic description of a common and extensible transactional workflow model which was specified in our previous works [10], [11]. Basically, within transactional workflow models, we distinguish between the control flow and the transactional behavior.

Basically, the transactional behaviour is observed in case of activity failures and defines recovery mechanisms supporting the automation of failure handling during runtime. Recovery mechanisms are used to ensure workflow fault-tolerance which is defined as the property allowing a business process to respond gracefully to expected but unusual situations and failures (also called exceptions). After an activity execution failure, it has to be decided, whether an inconsistent state was reached. According to this decision either a recovery procedure has to be started or the process execution continues according to the control flow (if the activity failure does not affect the execution of the workflow instance). Recovery mechanisms which are specified by transactional dependencies allow failed workflow instances to rollback to a coherent state if an inconsistent state was reached after activity failures. The goal is to recover the execution from a semantically acceptable state. Thus, the incoherent failure state can be corrected and the execution can be restarted from a coherent point thanks to the recovery mechanism. A coherent point is an execution step of the workflow (equivalent to a save point in database transactions) which represents an acceptable intermediate execution state. It is also a decision point where certain actions can be taken to either solve the problem that caused the failure or to choose an alternative execution path to avoid this problem [12]. Designers define according to their business needs for each failed activity the localization of the coherent point (for instance, the coherent after CA failure is located after CRS). As such, exception handling is part of the business logic of a workflow and may dominate its normal behavior [13].

1) Control flow: The control flow (or skeleton) of a workflow specifies the partial ordering of component activity activations. Within a workflow instance where all activities are executed without failure or cancellation, the control flow defines activity dependencies. In order to enhance reusability and common comprehension, we use the workflow patterns [14] as an abstract description of a reoccurring class of dependencies to describe the control flow as a pattern composition. We emphasize on the following seven patterns *sequence*, *AND-split*, *OR-split*, *XOR-split*, *AND-join*, *OR-join*, *XOR-join* and *OR-join* to explain and illustrate our approach.

2) Transactional Behavior: Basically, the transactional behavior is described through the activity's transactional properties and the transactional flow depicting respectively the intra and the inter activity *transactional dependencies* after activity failures. The *transactional dependencies* play an important role in coordinating and executing a workflow instance. These dependencies define the relationships that may exist between the events reporting activity execution state (canceled, failed, and terminated) of one or two activities specifying respectively the intra-activity state dependencies (i.e activity transactional properties) or the inter-activity state dependencies (i.e transactional flow).

- **Activity transactional properties :** Every activity can be associated to a life cycle state chart that models the possible states through which the executions of this activity can go, and the possible transitions between these states. These transitions describe the intra-activity state dependencies on which depend the activity transactional properties. The main transactional properties that we are considering are *retriable* and *pivot* [15]. An activity a is said to be retriable (a^r) *iff* it is sure to complete successfully. Even if it fails, a^r is reactivated until successful execution. a is said to be pivot (a^p) *iff* once the activity successfully completes, its effects remain and cannot be semantically undone or reactivated. A pivot activity cannot be compensated.
- **Transactional flow :** The transactional flow specifies the inter-activity state dependencies after activity failures. We distinguish two different transactional dependencies in the transactional flow: *alternative* dependencies, and *cancellation* dependencies. After an activity failure, a recovery process can be initialized by an alternative dependency that activates another activity, which is located through a related coherent point, to resume instance execution. This recovery mechanism can be either a backward recovery if the coherent point is located before the failed activity, or a forward recovery if the coherent point is located after the failed activity. An activity failure can cause also a cancellation (non-regular or abnormal end) of one or more active activities which is described through a cancellation dependency. This dependency can be enacted within the recovery mechanism in order to regain a consistent state. Indeed, the cancellation of running activities, after the activity failure, can avoid their undesired terminations.

3) Transactional constraints: The transactional behaviour depends on the control flow described through a set of transactional constraints. Indeed, the specification of a recovery mechanism defined through the transactional behaviour should respect transactional constraints which are partially dependent on the control flow to ensure reliable execution. Indeed, the recovery mechanisms (defined by the transactional flow) depend on the process execution logic (defined by the control flow). For example, regarding our motivating example, it was possible to define CC as an alternative to CH because (according to the XOR-split control flow operator) they are defined as exclusive branches. Similarly, it was possible to define a cancellation dependency from CCA to CIC because (according to the AND-join control flow operator) the failure of CCA requires the cancellation of any activity executed in parallel before recovering the workflow execution.

Concretely, these transactional constraints are inspired by [10] where "*potential*" transactional dependencies of workflow patterns are specified and proven. They determine the set of impossible transactional flows and then induce the set of "*potential*" transactional dependencies that can be defined according to a given pattern. The designer chooses, from this set, the effective transactional behaviour of the pattern that will be implemented and executed as the recovery mechanism in the event of the failure of one of the pattern's activities. Actually, there are three main rules : (i) a cancellation dependency can occur only between concurrent activities, (ii) we cannot have an alternative to concurrent flows, and (iii) no backward recovery can happen for pivot activities. These rules will be further developed and formally specified in section IV.

B. Petri nets and Wf-nets

In this section, we recall the definition of a Petri net and some basic notions of its theory.

Definition 3.1 (Petri nets): Let P and T be disjoint sets of places and transitions respectively, the elements of $P \cup T$ are called nodes. A Petri net is a tuple $N = \langle P, T, Pre, Post \rangle$ with the backward and forward incidence matrices Pre and $Post$ defined by Pre (resp. $Post$) : $(P \times T) \longrightarrow \mathbb{N}$. We denote by $Pre(t)$ (resp. $Post(t)$) the column vector indexed by t of the matrix Pre (resp. $Post$). The preset of a place p (resp. a transition t) is defined as ${}^\bullet p = \{t \in T \mid Post(p,t) > 0\}$ (resp. ${}^\bullet t = \{p \in P \mid Pre(p,t) > 0\}$), and its postset as $p^\bullet = \{t \in T \mid Pre(p,t) > 0\}$ (resp. $t^\bullet = \{p \in P \mid Post(p,t) > 0\}$).

Definition 3.2 (Marking and transition enabling): A marking of a net is a mapping $P \longrightarrow \mathbb{N}$. We call $\langle N, m_0 \rangle$ a net system where N is a Petri net and m_0 its initial marking. A marking m enables the transition t ($m \xrightarrow{t}$) if $m(p) \geq Pre(p,t)$ for each $p \in {}^\bullet t$. In this case the transition can occur (or be fired), leading to the new marking m' that is given by $m' = m - Pre(t) + Post(t)$. We denote this occurrence by $m \xrightarrow{t} m'$. The *reachability marking graph* of a Petri net contains nodes for each marking m and an arc from m to m' labeled by t if $m \xrightarrow{t} m'$. If there exists a chain ($m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \rightarrow \ldots \xrightarrow{t_n} m_n$), then we say that the sequence $\sigma = t_1 \ldots t_n$ is firable from m_0, denoted by $m_0 \xrightarrow{\sigma}$, while the reachability of m_n from m_0 by firing σ is denoted by $m_0 \xrightarrow{\sigma} m_n$. Also, the sequence σ is called a computation of the Petri net N. We denote by T^* the set of finite sequences of T. The finite language of (N, m_0) is the set $L^*(\langle N, m_0 \rangle) = \{\sigma \in T^* \mid m_0 \xrightarrow{\sigma}\}$ and $[N, m_0\rangle = \{m$ s.t. $\exists \sigma \in T^*,\ m_0 \xrightarrow{\sigma} m\}$ represents the set of reachable markings of $\langle N, m_0 \rangle$.

Definition 3.3 (Language projection): Let σ be a sequence of transitions ($\sigma \in T^*$). The projection of σ on a set of transitions $X \subseteq T$ (denoted by $\sigma_{\lfloor X}$) is the sequence obtained by removing from σ all transitions that do not belong to X. The projection function is extended to sets of sequences (i.e. languages) as follows: $L_{\lfloor X} = \{\sigma_{\lfloor X},\ \sigma \in L^*\}$.

Definition 3.4 (Wf-net): To model workflows, we use workflow nets (Wf-nets) [16]. Let $N = \langle P, T, Pre, Post \rangle$ be a Petri net. N is said to be a workflow net (Wf-net) if

- there is one source place $i \in$ P s.t. ${}^\bullet i = \emptyset$ and one sink place $o \in$ P s.t. $o^\bullet = \emptyset$, and
- every node $x \in P \cup T$ is on a path from i to o.

Note: As far as the behavior of a workflow is concerned, the corresponding Wf-net is associated to an initial marking where only the source place is marked with a single token. Besides, a *final marking* of Wf-net is every one, that is reachable from a such initial marking, where the sink place is marked. Thus, given a Wf-net W and an initial marking m_0 associated with W, the language of a (W, m_0) is now defined as follows: $L^*(\langle W, m_0 \rangle) = \{\sigma \in T^* \mid m_0 \xrightarrow{\sigma} m_f\}$ where m_f is a final marking.

C. LTL model checking

Model checking for Linear TemporaLogic (LTL) [17] is usually based on converting the (negation of a) property into a Büchi automaton, composing the automaton and the model, and finally checking for emptiness of the language of the composed system. The last step is the crucial stage of the verification process because of the state explosion problem. LTL is built up from a set of propositional variables, the usual logic connectives $\wedge$, $\vee$, $\neg$, $\implies$ and the following temporal modal operators: X for next, $\mathcal{G}$ for always (globally), $\mathcal{F}$ for eventually (in the future) and $\mathcal{U}$ for until. Reader can see [17] for detailed description of the syntax and semantic of LTL.

Since LTL is interpreted on infinite paths, a usual solution to check LTL formula on a system is to convert each of its finite maximal paths to an infinite one by adding a loop on its dead states. Here maximal paths are those ending on final markings.

IV. Formal specification of a transactional workflow

In this section, we show how Petri nets and LTL logic can be combined in order to specify correct behavior of transactional workflows. If the description of the control flow part of a transactional workflow can easily be obtained using existing approaches (e.g. [14] or [18]), the modeling and especially the verification of the transactional behavior is a difficult process. This section is mainly dedicated to this issue. Each activity transactional property and each dependency between two activities is modeled by a Petri net completed by some LTL formulae that have to be satisfied by the model so that the expected transactional behavior is respected.

A. Modeling the control workflow

The WF-net associated with a control flow can be easily obtained by representing each activity A by a Petri net transition A, which firing means the activation of A, completed with a couple of places ($Enable_A$, $After_A$) as input and output places of A, respectively. Some basic rules can then be applied in order to obtain the whole WF-net. For instance, fusion of the input and output places of two successive activities, fusion of the input places of alternative (exclusive choice) activities ... etc. The reader can refer to [14] or [18] for details about the modeling of control workflow patterns using Petri nets.

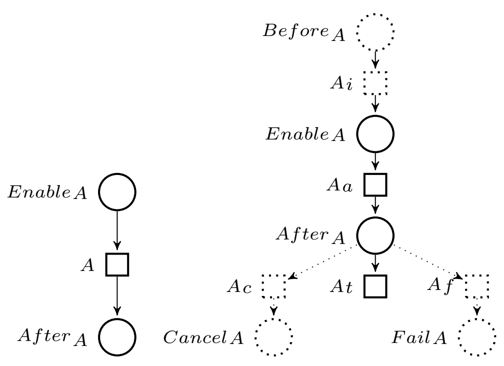

Figure 2. Petri net representation of critical and non critical activities

B. Modeling the transactional flow

In this section, we supply some Petri net frames to model the transactional behaviors of workflow activities. For each transactional property (resp. dependency), a Petri net pattern is proposed associated with LTL formulae that have to be satisfied by the model in order to ensure the consistency of these properties (resp. dependencies). We distinguish the two following cases:

1) For transactional properties (i.e. pivot and retriable), we propose a Petri net structure guaranteeing a consistent behavior w.r.t. the specification. Alternatively, we express the desired behavior using LTL formulae that can be checked on the transactional workflow.
2) For transactional dependencies (i.e. alternative and cancellation), each constraint (structural and semantic) is expressed by an LTL formula. A Petri net pattern is also proposed such that the semantic constraints are structurally satisfied (i.e. by construction of the Petri net). Since the structural constraints cannot be proved locally to such a pattern (we need to have the

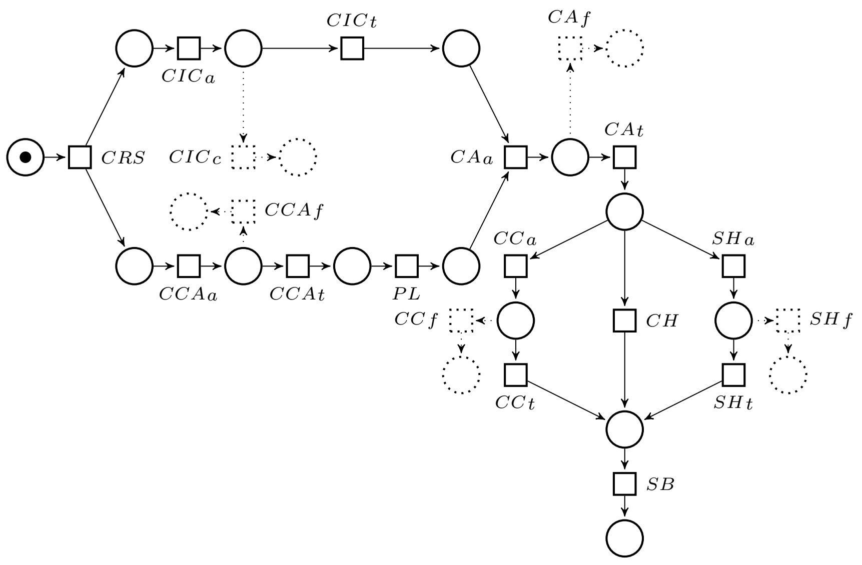

Figure 3. Control flow with critical activities

whole model), the corresponding LTL formulae must be checked on the complete transactional workflow using any existing model checking tool. For instance, the structural constraint : "an alternative dependency between two activities A and A' is not authorized if A and A' are concurrent (parallel) activities" can not be checked on the Petri net pattern. We will propose to model such a dependency (where only A and A' are present) . Conversely, the semantic constraint of the alternative dependency ("when A fails then it will enable A'") can be ensured structurally and locally to the proposed Petri net pattern as we will see in the following.

Before we attack the formal specification of activities transactional properties and transactional dependencies, let us fix some hypothesis: Once the control flow is specified and modeled by a WF-net, one can slightly change the model by considering the critical aspect of each activity. Non critical activities are still modeled (like in the control flow) by a simple Petri net structure (Figure 2, left hand side). However, critical activities, which can fail, be canceled or both, will be represented as illustrated in Figure 2, right hand side. When A could fail (resp. be canceled), then the execution of A is split into two steps: First A is activated, which is modeled by the transition A_a (with $Enable_A$ and $After_A$ as input and output place respectively). Then, the result of the activation of A is to be known, either it is a success or a failure (resp. a cancellation). These two possible results are represented by two transitions A_t, for the success of A and A_f (resp. A_c) for failure (resp. cancellation) of A. Moreover, the dotted place $Before_A$ and transition A_i can be added in the following if we need to represent the state of the flow before enabling the activity A.

Using the previous translation process and the information about the critical activities in our motivating example, the control flow can be represented by Figure 3. The modification w.r.t. to Figure 1 is represented by dotted nodes and arcs.

Now, the main difficult task for designers is to take in account the transactional dependencies (intra and inter activities) at the model level so that it fulfills the desired behavior. In the following, we will help designers in this process using Petri net and LTL logic formalisms.

C. Activity transactional properties

1) Retriable dependency: Given an activity A, A is said to be *retriable* if and only if, when it fails then it can be reactivated until it succeed in the future. We distinguish two possible interpretation for the *retriable* property: a relaxed interpretation and a strict interpretation. In the first one, A is retriable if: when A fails, then it is always possible in the futur for A to ends successfully. The strict interpretation says that: when A fails then it always ends sucessfully in the future.

The relaxed interpretation can be ensured structurally as described by the left hand side of Fig 4.

Conversely, the strict interpretation cannot be ensured structurally because of the indeterministic feature of transitions firing in a Petri net. However, one can propose the structure of the right hand side of Figure 4 where the success of the retriable activity A is forced after, at most, N failures of A. The maximum number of successive failures N is assumed to be known a priori.

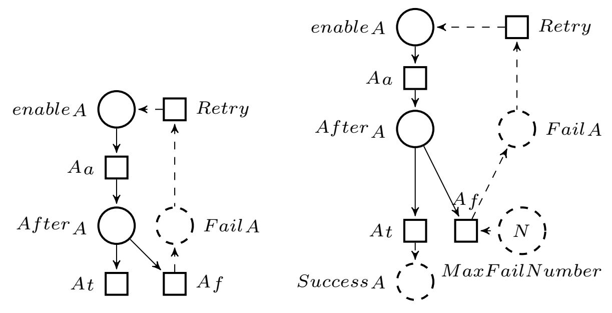

Figure 4. Relaxed and strict retriable rule

The relaxed and strict retriable property can now be expressed by two LTL formulae respectively : $G(Fail_A \implies \mathcal{F}After_A)$ and $G(Fail_A \implies \mathcal{F}\ Success_A)$. It is clear that these formulae are satisfied by left hand and right hand side of Figure 4.

2) Pivot activity: Given an activity A, A is said to be *pivot* if and only if, when it is activated, A can never be reactivated in the future. Figure 5 represents a Petri net ensuring structurally such a property. The pivot property can be expressed with the following LTL formula: $After_A \implies X(\mathcal{G}\ (\neg Enable_A))$. The proposed Petri net structure ensures this formula since the added place P, initially marked, will never be marked as soon as A is activated.

Note: The *pivot* and *retriable* properties are incompatible i.e. an activity A cannot be pivot and retriable at once.

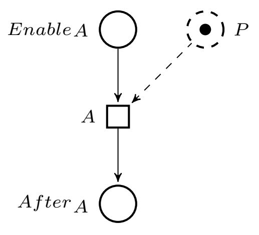

Figure 5. Pivot activity

D. Transactional dependencies

1) Cancellation dependency: The cancellation dependency between two activities A en A' is an asymmetric relation: A cancels A' means: if A fails while A' is enabled, then A' should either fail or be cancelled. Such a dependency implies the following structural constraint: Activities A and A' should belong to two parallel (sub-) flows.

The cancellation dependency and the corresponding structural constraint can be expressed using the two following LTL formulae respectively:

1)
$$\mathcal{G}(Fail_A \wedge Enable_{A'} \implies \mathcal{F}\ After_{A'} \wedge X(\neg Success_{A'})$$

2)
$$\mathcal{F}(Enable_A \wedge Enable_{A'})$$

Figure 6 gives a Petri net modeling the cancellation dependency that ensures the satisfaction of the first property. Mainly, two places and two transition have been added w.r.t. the WF-net associated with the control flow. The place denoted by $Nenable_{A'}$ (standing for A' is not enable) is added as a complementary place of $Enable_{A'}$ (i.e., $M(Nenable_{A'}) + M(Enable_{A'}) = 1$ for all reachable marking M) , and the Exc place is added in order to forbid the "normal" termination of A' activity (i.e. firing of A'_t) as soon as the A activity fails (A_f fired). Both places are initially marked with one token. Once the A activity fails (i.e. the A_f transition is fired), the two transitions $C_{A'}$ and OK allow to cancel A' or not respectively.

Regarding to the structural constraint, one cannot check it without having the whole model. It has to be checked afterward and it ensures the concurrent behavior between two activities (or two Petri net transitions) in a safe model (i.e. 1-bounded).

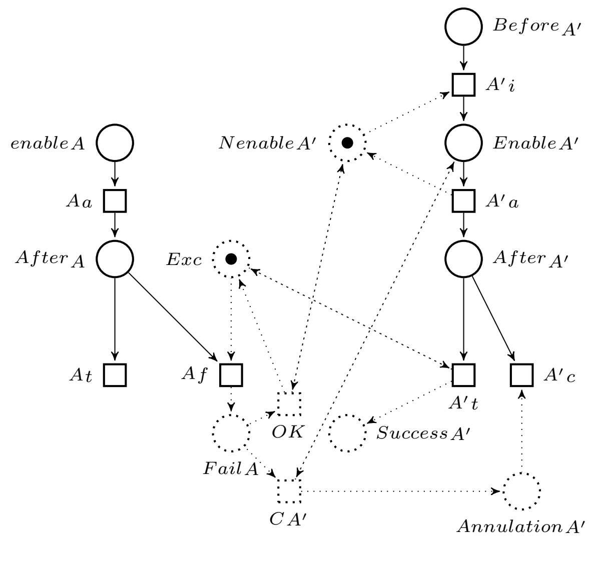

Figure 6. Cancellation dependency

2) Alternative dependency: We say that an activity A' is an alternative to an other activity A when A' should be activated whenever A fails. Such a dependency is prohibited between parallel activities. Thus A and A' should be either exclusive (never simultaneously enabled) or ordered (sequential). Figure 7 shows how to complete the control flow so that the first condition holds. These two properties can be expressed using the two following LTL formulae respectively: $\mathcal{G}(Fail_A \implies Enable_{A'})$ and $\mathcal{G}(\neg(Enable_A \wedge Enable_{A'}))$

While the first formula is ensured by the Petri net structure, the second one can not be checked locally and depends on reminding part of the model. It can be checked afterward using any existing LTL model checker.

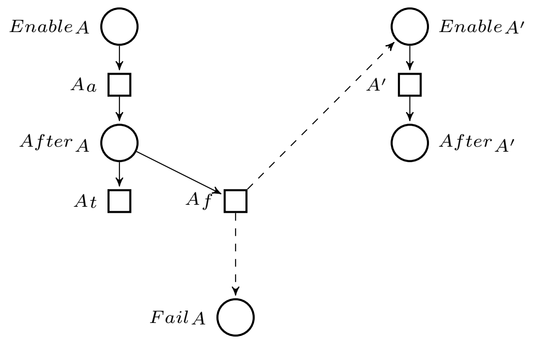

Figure 7. Alternative dependency

Depending on the order between A and A' the alternative recovery can be qualified by *forward recovery* (A' after A) or *backward recovery* (A' before A).

1) **Forward recovery**
In this case, an additional structural constraint must hold. It says that activities A and A' are ordered (belong to a sequential flow) and that A' is always activated after A. This can be expressed with the following LTL formula: $\mathcal{G}(After_A \implies X(\neg After_A \mathcal{U} After_{A'}))$
2) **Backward recovery dependency**
The backward recovery is dual to forward recovery. The additional structural constraint should satisfy: A' is always activated before A. The associated LTL formula : is $\mathcal{G}(After_{A'} \implies X(\neg After_{A'} \mathcal{U} After_A))$

E. Application

Applying the approach described in the previous section on our car rental example leads to the Petri net of Figure 8. The verification of this model has been done using an LTL model checking prototype [19] based on binary decision diagrams [20]. It is an on-the-fly model checker based on the emptiness check of the synchronized product of the model and the automaton of the (negation of the) formula to be checked. The expected behavior of transactional property has been checked as well as the structural constraints.

We also, tried to introduce some faults in the model (for instance a cancellation dependency between the CH and CC activities or an alternative dependency between CIC and CCA) and check that the corresponding LTL formulae are not satisfied.

V. Related works

Some studies investigated the issues related to the exception handling and recovery from activity failures in workflow management systems. A first discussion on workflow recovery issues has been presented in [21]. The necessity of workflow recovery concepts was partly addressed in [3]. Especially, the concept of business transactions, introduced in [22], deals with some basic workflow recovery. A significant amount of work applies the concepts introduced in transaction management to business process environments. Hamadi et al. [23] offer an exhaustive survey. Besides, they describe a framework for the specification of high-level recovery policies that are incorporated either with a single task or a set of tasks, called Recovery Regions. They propose an extended Petri net model for specifying exceptional behavior in business processes and adapt the mechanisms of the underlying Petri net at run time to handle exceptions while keeping the Petri net design simple and easy.

In this paper, we have presented a transactional workflow model that should be considered as a support model to our verification approach which are the paper's main contribution. Basically, the described transactional behavior of this workflow model is common to existing transactional workflows model. It is specified by a set of activities, the dependencies between these activities, and the associated recovery mechanisms. Our workflow transactional model is flexible enough to include additional recovery mechanisms. Indeed, the considered transactional behavior is common and can be enriched to include other rules dependent on business semantics. Therefore, our model can be customized for different kinds of transaction models by restricting or extending the type of dependencies that can exist between the activities. This customization is done by limiting or extending the types of the inter or intra activity dependencies. Thanks to this flexibility, the extension or the adaptation of the above described transactional workflow model can be easily deployed with minor changes. However, adding new policies may provide support and flexibility, but it also makes the business process model more complex. Complexity severely compromises the usability and adoption. Therefore, our proposed transactional workflow model tries to remain in a striking balance between expressive power and simplicity.

It should be noted that our focus in this paper is not on specifying a new transactional workflow model, but on a comprehensive description of activity-based recovery policies that are used in the following as a support model for our verification and correction techniques. Recently, we have specified in [24] a combined approach that describes a formal framework, based on Event Calculus and on theorem prover to check the transactional behavior of composite services before and after execution. Our approach provides a logical foundation to ensure transactional behavior consistency at design time and report recovery mechanisms deviations after runtime. Compared to [24], our Petri net driven verification approach presented in this paper is a more natural candidate for process verification and modeling. Indeed, Petri net is definitely the most popular process modeling language. Therefore, using Petri net ensures that

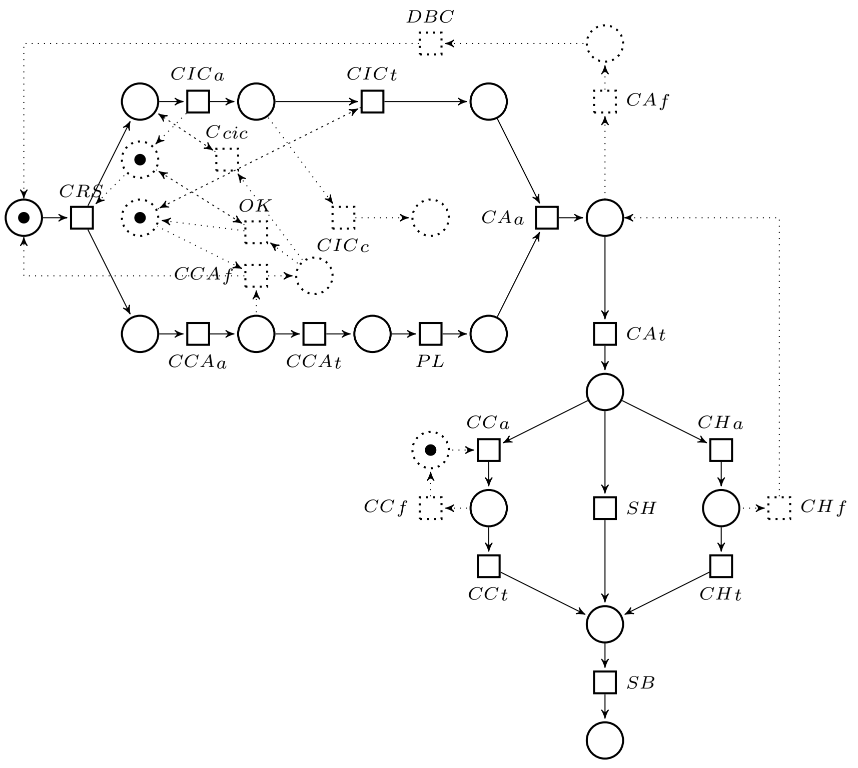

Figure 8. Correct transactional flow

our approach is closer to process designers considerations and habits. Moreover, [24] requires a process model transformation to Event Calculus which is not needed in our approach as the workflow models are often described in Petri nets. Finally, theorem proving based verification used in [24] is well known to be not fully automatic and may involves the intervention of the user to help the proof while LTL model checking is completely automatic.

VI. Conclusion

Workflow management systems are increasingly deployed to deliver reliable e-business transactions across organizational boundaries. To ensure a high service quality in such transactions, failure handling re-engineering for reliable executions is needed. However, failed instance executions may arise because it is hard to predict workflow transactional behavior that can cause an unexpected activity failure. In this paper we have presented an original approach to ensure reliable workflow transactional behavior. Our work attempts to apply verification techniques to check model correctness and correct design errors. We described in this paper a combined approach that describes a formal framework to check workflow transactional behavior. Our approach provides a logical foundation to ensure transactional behavior consistency at design time and report recovery mechanisms deviations. The transactional workflow formally specified (using LTL) as a set of logical formulas are used thereafter in our paper to support reasoning about its recovery mechanisms to check its transactional consistency.

An immediate perspective of this work is to implement the approach so that the designer has only to specify his model and enumerate the properties and dependencies to be respected. The aimed tools will then proceed to the modeling and the verification of the associated behaviors automatically. The output is either the validation of the model or help again the designer to detect the error (by given counter examples) and repeat the process until the model is proved correct. We are also working on enhancing the capabilities of the transactional behavior monitoring techniques by extracting data flow dependencies. We aim also to adapt/combine verification techniques to the web services related fields.

References

[1] J. Veijalainen, F. Eliassen, and B. Holtkamp, "The S-transaction Model," in *Database transaction models for advanced applications*, A. Elmagarmid, Ed. Morgan Kauffman, 1990.

[2] U. Dayal, M. Hsu, and R. Ladin, "Business process coordination: State of the art, trends, and open issues," in

VLDB, P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, Eds. Morgan Kaufmann, 2001, pp. 3–13.

[3] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: from process modeling to workflow automation infrastructure," *Distrib. Parallel Databases*, vol. 3, no. 2, pp. 119–153, 1995.

[4] B. Kiepuszewski, R. Muhlberger, and M. E. Orlowska, "Flowback: providing backward recovery for workflow management systems," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. ACM Press, 1998, pp. 555–557.

[5] W. M. Van Der Aalst and A. H. M. Ter Hofstede, "Verification of workflow task structures: A petri-net-based approach," *Inf. Syst.*, vol. 25, no. 1, pp. 43–69, 2000.

[6] J. Siegeris and A. Zimmermann, "Workflow model compositions preserving relaxed soundness.." in *4th International Conference on Business Process Management*, ser. Lecture Notes in Computer Science, vol. 4102. Vienna, Austria: Springer-Verlag, 2006, pp. 177–192.

[7] A. Martens, "Simulation and Equivalence between BPEL Process Models," in *Proceedings of the Design, Analysis, and Simulation of Distributed Systems Symposium (DASD'05), Part of the 2005 Spring Simulation Multiconference (SpringSim'05)*, San Diego, California, Apr. 2005.

[8] K. Klai, S. Tata, and J. Desel, "Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes," in *BPM*, 2009, pp. 294–309.

[9] A. Sheth and M. Rusinkiewicz, "On transactional workflows," *Special Issue on Workflow and Extended Transaction Systems IEEE Computer Society , Washington DC*, 1993.

[10] S. Bhiri, O. Perrin, and C. Godart, "Extending workflow patterns with transactional dependencies to define reliable composite web services." in *AICT/ICIW*. IEEE Computer Society, 2006, p. 145.

[11] W. Gaaloul, S. Bhiri, and A. Haller, "Mining and reengineering transactional workflows for reliable executions," in *ER*, ser. Lecture Notes in Computer Science, C. Parent, K.-D. Schewe, V. C. Storey, and B. Thalheim, Eds., vol. 4801. Springer, 2007, pp. 485–501.

[12] W. Du, J. Davis, and M.-C. Shan, "Flexible specification of workflow compensation scopes," in *Proceedings of the international ACM SIGGROUP conference on Supporting group work : the integration challenge*. ACM Press, 1997, pp. 309–316.

[13] J. Moss, "Nested transactions and reliable distributed computing," in *Proceedings Of The 2nd Symposium on Reliability in Distributed Software and database Systems*. IEEE CS Press, 1982.

[14] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski, "Advanced Workflow Patterns," in *5th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'00)*, ser. LNCS, O. Etzion and P. Scheuermann, Eds., no. 1901. Eilat, Israel: Springer-Verlag, September 6-8, 2000, pp. 18–29.

[15] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz, "A multidatabase transaction model for interbase," in *Proceedings of the sixteenth international conference on Very large databases*. Morgan Kaufmann Publishers Inc., 1990, pp. 507–518.

[16] W. Aalst, "The Application of Petri Nets to Workflow Management," *The Journal of Circuits, Systems and Computers*, vol. 8, no. 1, pp. 21–66, 1998. [Online]. Available: citeseer.ist.psu.edu/vanderaalst98application.html

[17] A. Pnueli, "The temporal logic of programs," in *FOCS*, 1977, pp. 46–57.

[18] W. M. P. van der Aalst and T. A. H. M. Hofstede, "Workflow patterns: On the expressive power of (petri-net-based) workflow languages," in *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, ser. DAIMI, K. Jensen, Ed., vol. 560, August 2002, pp. 1–20.

[19] K. Klai and D. Poitrenaud, "Mc-sog: An ltl model checker based on symbolic observation graphs," in *Petri Nets*, ser. Lecture Notes in Computer Science, K. M. van Hee and R. Valk, Eds., vol. 5062. Springer, 2008, pp. 288–306.

[20] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, 1992.

[21] W. W. Jin, M. Rusinkiewicz, L. Ness, and A. Sheth, "Concurrency control and recovery of multidatabase work flows in telecommunication applications," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. ACM Press, 1993, pp. 456–459.

[22] F. Leymann, "Supporting business transactions via partial backward recovery in workflow management systems," in *Proceedings of BTW?95*. Springer, 1995, pp. 51–70.

[23] R. Hamadi, B. Benatallah, and B. Medjahed, "Self-adapting recovery nets for policy-driven exception handling in business processes," *Distributed and Parallel Databases*, vol. 23, no. 1, pp. 1–44, 2008.

[24] W. Gaaloul, M. Hauswirth, M. Rouached, and C. Godart, "Verifying composite service recovery mechanisms: A transactional approach based on event calculus," in *15th International Conference on Cooperative Information Systems CoopIS07*. Springer-Verlag, November, 2007.