Lookahead Consistency Models for Dynamic Migration of Workflow Processes

Ahana Pradhan and Rushikesh K. Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay, Powai, Mumbai-400076, India

Abstract. Dynamic migration of workflows requires the notion of consistency for safe migration. The literature primarily covers consistency models based on the history of the workflow to be migrated. However, for several situations, the history based models are not enough to decide migratability of a state. The paper introduces lookahead models of consistency, which are based on the question of how the remaining part of the workflow is treated in the new process. Three lookahead models are described and are illustrated with the help of example cases of realistic migration scenarios. Moreover, in certain situations, even if there is a consistent lookahead migration possible, the tokens can not be directly migrated into the new net due to contradictory traces available in the new net. The paper also proposes an algorithm called *Accept-Reject Branching* to compute the contradictory segment of the new net. A detail case study of a library resource acquisition workflow is presented to highlight the contributions.

Keywords: Workflow Migration, Consistency, Lookahead.

1 Introduction

In an ever-changing business environment the business processes do not remain static. Instances of one workflow often need to follow the evolved business logic of a different, or a refined schema. When an instance migrates from the old process into the new process the issue of consistency has to be addressed. The notion of consistency ensures that a migrated instance finishes execution without encountering a runtime error or inconsistencies in application semantics.

One of the initial works discussing consistency in dynamic evolution of workflows is by Ellis et al. [1]. They describe consistency criterion as the possibility of reproducing the execution history in the new schema. Moreover, mapping the old history in the new schema obtains the runtime state of the workflow from where it can resume to follow the new business logic. In this approach the consistency is based only on the past execution of the old instance. Contemporary works by Casati et al. [2], and Sadiq et al. [3] also adopt the same notion of consistency under the terminology of *compliance*. In the context of instance-specific ad-hoc dynamic changes, Reichert et al. [4] present the notion of consistency as a criteria that preserves the validity of the instance-specific workflow schema after

modification and suits the old execution history. Later researchers have adopted this same notion of history equivalence consistency under different terminologies and with subtle differences in the interpretation of history. The notions of *valid mapping* in the work of Weske [5], *migration conditions* by Dias et al. [6], several classes of *compliances* by Rinderle et al., [7], [8], by Sun and Jiang [9], and the *consistency criteria* in our previous work [10] are examples of history-based consistency.

Notable works using Petri net models of workflows, on the other hand, adopt the notion of consistency defined on the basis of marking. In Petri net models various process states are explicitly modeled by places. In this model a marking represents the current state of the process. Consistent markings in the old and the new workflow are decided based on the equivalence of states. This approach is taken in the works by Van der Aalst [11] and later by Circirelli et al. [12].

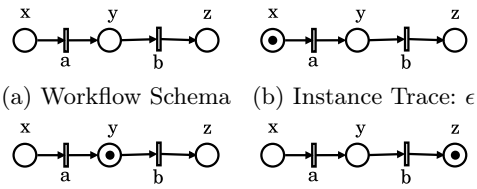
History based consistency model derives its motivation in the need to consider as done what is already accomplished and proceed exactly thereafter by resuming the workflow as per the new logic. Therefore, a primary goal of interest in history based migration is to ascertain the preconditions before migration takes place. However, apart from the models of history and state based consistency, dynamic workflow migration in the context of other business goals such as resource optimization, incidental migration, and handling eventualities often require a lookahead consistency criterion. Motivating examples of such dynamic migration scenarios are presented in this paper, on the basis of which, we develop the notion of lookahead consistency. A Petri net based modeling notation called WF-net [13] is used for representing workflows. The consistency models are defined using the WF-net notation, which can also be adopted in generic workflow terminology in a model independent manner. Three variants of lookahead consistency called strong, accommodative and weak lookahead are introduced.

An algorithm for generating weak lookahead consistent migrations for workflow instances is presented. It is possible that newer paths may be available in the new net for a migrating instance. In order to enforce a stricter lookahead, these paths need to be blocked, which can be done by blocking the head-transitions of the contradictory segments. These blocking transitions are identified using an algorithm called *accept/reject branching*.

The paper is organized as follows. Section 2 briefly outlines the preliminaries of Petri net based workflow model. Section 3 discusses the related work and the contributions of the paper. Sections 4, 5 and 6 discuss the three lookahead consistency models with the help of realistic application scenarios. Section 7 lastly discusses the algorithms for lookahead consistency along with a case study.

2 Notations

In this section a brief background is provided for WF-net notation of Van der Aalst [13], which is used in this paper as a reference formal notation for defining consistency and for developing the case studies.



(a) Workflow Schema (b) Instance Trace: ϵ

(c) Instance Trace: **a** (d) Instance Trace: **a,b**

Fig. 1. WF-net Model of Workflow and Workflow Instances

Fig. 1a depicts an example WF-net schema of a sequence workflow involving two tasks **a** and **b**. Tasks are represented as *transitions* (rectangles), and conditions are represented as *places* (circles). A place models the precondition and postcondition of its immediate successor and predecessor tasks respectively. The *source place* captures the initial condition i.e. the start condition that triggers the start of the corresponding workflow. Similarly, the *sink place* models the terminal condition. A *token* (a dot in a place) makes up a *marked place*. If all pre-conditions of a transition are true, the transition is *enabled*. An enabled transition eventually *fires* by consuming one token from each of its pre-places (all pre-conditions), and produces one token in each of its post-places (post-conditions). A postcondition can be satisfied by firing of any one of its preceding transitions.

A placement of tokens in a net is called a *marking*. A marking represents the runtime state of an instance of a business process modeled by the net. A *firing* of a transition changes the marking. A *firing sequence* is a sequence of transition firings from one marking to another. The firing sequence from initial marking to the marking shown in Fig. 1b is empty. Fig. 1c shows the marking after firing of transition **a**, i.e. after completion of activity **a**, where the state of the workflow is *y*. Similarly, Fig. 1d shows the final marking of the workflow after firing of **b**.

The following notations are used to represent the dynamics of the net: A unit transition of marking $m_1 \xrightarrow{\mathbf{t}} m_2$ means that firing of transition **t** changes the marking (state) of the net from m_1 to m_2. In general a transition $m_1 \xrightarrow{\sigma} m_2$, where σ is a firing sequence $t_1 t_2 ... t_n$ represents that the particular firing sequence σ changes the state of the workflow from m_1 to m_2 through some other states. Multiple firing sequences between two markings may be possible in nets with choice and concurrency. In the figure, $x \xrightarrow{\mathbf{a}} y$ is an example unit transition, and $x \xrightarrow{\mathbf{ab}} z$ is a transition from x to z through a longer firing sequence.

3 Related Work and Contributions of the Paper

The literature includes a variety of consistency criteria in the context of runtime migration of workflows [14], [7]. They are primarily history and state based approaches. For instance, the state based approach of *behavioral consistency* in the work of Casati et al. [2] looks into validity of the mapped state to ensure proper termination. Consequently their approach does not need to take the actual content and its variations in possible future execution traces into account.

An example of history based consistency is the approach of Ellis et al. [1], in which, the migrated instance state is required to have, starting from the initial marking, the same trace that was before migration.

The only work in this context, where the possible lookahead parameters were taken into account is the notion of *inheritance of workflows* by Van der Aalst and Basten [15]. In their approach, consistency is defined based on *branching bisimilarity* between the states of the old and the new workflows. As per the notion of bisimulation [16], equivalence is established between two states based on their future transitions and the states visited by those transitions. Branching bisimilarity considers inclusion of silent transitions in addition. Therefore, the adopted model of consistency in this work falls into the class of lookahead based model. However, adopting the notion of bisimulation defined on LTS (labeled transition system) states leads to much stronger criteria than intended in the context of process migration. The addressed domain of process migration does not consider a process to be interactive. In particular, the problem of dynamic instance migration addressed in this paper does not consider conversation or collaboration issues. Therefore, observational behavior of a process is its *trace*, and hence, equality relations based on the traces are sufficient to define consistency. Our work takes up this point to develop a range of consistency models that are based on lookahead traces.

Consulting lookahead parameters in the context of dynamic web-service protocol evolution has been identified as a necessary feature in the work of Ryu et al. [17]. They describe the notion of *forward compatibility* that is a property to be considered in the context of migrating web-service conversations. The forward compatibility captures the ability for the clients to interact with the dynamically evolved service after migration without confronting an *error*. Therefore, in order to save the ongoing conversations from failing, the lookahead parameters are vital. However, in contrast with the web-service conversation situation, as pointed out in the previous paragraph, the dynamic migration in the context of workflows are rather the changes in orchestration itself. Consequently, the need for consideration of lookahead parameters requires a solution with focus moving from interaction error to consistency in application semantics.

A benefit of these newly introduced models is that they can be used independently or in commune with the history based or state based consistency models as per the need of a particular workflow migration scenario. A lookahead based migration approach can then be applied considering the traces starting from the current state in the old workflow in the migration pair, and finding them in the new workflow to compute consistent migration.

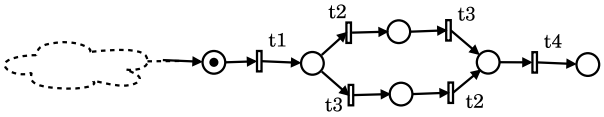
The proposed lookahead models are demonstrated with the help of motivating cases. In the subsequent sections we define and illustrate three lookahead consistency models, which are *strong lookahead consistency*, *accommodative lookahead consistency* and *weak lookahead consistency* models respectively. The accommodative and the strong lookahead consistency models are specializations of the weak model, and the strong model is a specialization of the accommodative model.

4 Strong Lookahead Consistency

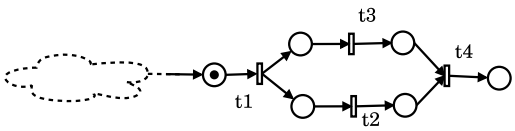
In this model, consistent states in the old and the new workflows are mapped by equivalence between possible futures in both the workflows. States of the old and the new workflows are consistent if the schedules that are yet to be completed from the old workflow state are the only schedules that are possible in the new workflow after migration. The strong lookahead model does permit changes to the net as long as the set of possible future traces is the same. This model can be applied to handle migration situations in process re-engineering cases, in which, from the point of view of the current state of the migrating instance, a change should not be perceived as far as the traces, i.e. the choices and the sequences thereby are concerned. In practice, such a situation may arise due to maintenance and compatibility issues. One such situation is illustrated later through a case study of a food packaging workflow example.

The WF-net based definition of the strong lookahead consistency model is given below. We use relational operator $\blacklozenge$ between two markings to represent strong lookahead consistency between them. Subsequently relational operators $\ddot{\diamond}$ and $\diamond$ are used to represent the accommodative and the weak lookahead models.

Definition 1 *Let the old and the new workflows be modeled as WF-nets W and W' respectively, m_f and m'_f be the final markings in W and W' respectively, m be a marking in W, and m' be a marking in W'. Let $F_m = \{\sigma | m \xrightarrow{\sigma} m_f\}$, i.e. be the set of all firing sequences starting from marking m and reaching the final marking m_f in the old net. Similarly, $F'_m = \{\sigma' | m' \xrightarrow{\sigma'} m'_f\}$, i.e. be the set of all firing sequences starting from marking m' and reaching the final marking m'_f in the new net. Strong Lookahead consistency $m \blacklozenge m'$ is defined by the following trace equivalence condition: (i) $\forall \sigma \in F_m$, $\sigma \in F'_m$, and (ii) $\forall \sigma' \in F'_m$, $\sigma' \in F_m$. In other words, $m \blacklozenge m'$ is defined by the equality $F_m = F'_m$.*



Old Workflow Instance



New Workflow Instance in Optimized Schema

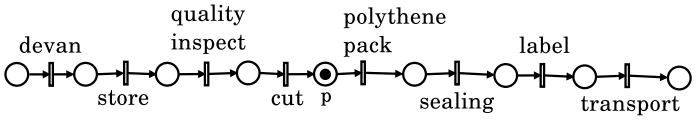
Fig. 2. Strongly Consistent Markings

It can be noted that to satisfy the above criteria, the structure of the downstream nets of the old and new workflows need not be the same. Intentionally different designs resulting into the same behavior may also fit to the definition. For example, a new net may be an alteration of its corresponding old net for achieving an optimization in the design as shown in Fig 2. In this example, the set of traces is $\{t_2t_3, t_3t_2\}$, which is same in both the nets. As the firings of transitions in Petri net semantics (reachability graphs[18]) are atomic and therefore instantaneous, a realization of parallel transitions t_2 and t_3 in the net results in two possible firing sequences t_2t_3 and t_3t_2. Therefore, since the sets of all possible traces from the shown markings are the same, they satisfy strong lookahead consistency.

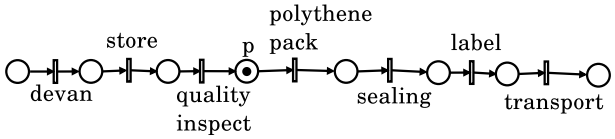
However, real-life workflows may have long duration tasks, in which, the semantics of non-overlapping atomic executions may not be possible. In such cases, the trace-based model can still be applied considering discrete events such as commencements or completions. For example, in an academic setup, if performing two courses in two consecutive semesters and performing them together in parallel in one semester needs to be considered as equivalent, it can be done so by considering the events marking the first lectures of the two courses. So, though there is physical interleaving of the task actions, if an academic process believes that the interleaving is acceptable as long as strong lookahead consistency is maintained w.r.t. the courses, a migration of a student from one system to another is possible. This assumption is useful in applying the lookahead models in non-Petri net workflow models such as ADEPT2 [19].

Fig. 3 shows packaging workflows for milk-products, chocolate and dry-fruits respectively. The packaging company imports the food items in bulk from various food processing companies and delivers them to distributors after packaging. The activities of devanning, storing items in the warehouse, quality inspection, and transport for delivery are manual activities. The fourth task in the dry-fruit packaging workflow is a manual task of inserting fruits by weight. Cutting of cheese and butter, food packaging in polythene or cardboard boxes, packet sealing and labeling are user assisted automated activities. As shown in the figure, in the case of dry-fruit, two kinds of packets are produced by the workflow using polythene packets or cardboard boxes. The workflow process is organized such that the packaging choice automatically alternates after regular intervals.

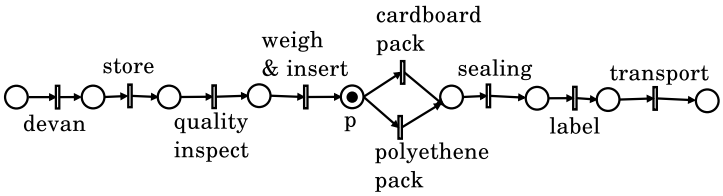
The packaging unit uses three different assembly-lines for the three food items. However, we can observe that polythene-based packaging of milk-products can use part of the assembly-line for the chocolate packaging once they reach the equivalent state (i.e. marking) p. Such an equivalent state is not available in the assembly-line of the dry-fruit packaging workflow due to the strictly alternating packaging designed feature. As the possible future execution sequences of the workflows for milk-products and chocolate packaging are exactly the same from the shown markings, a single assembly-line can take care of both of the packaging processes downstream the equivalent markings. The migration decision can be helpful when one assembly-line needs to be shut down for maintenance. Such a migration does not require any modification to the new process when strong



Workflow for Butter and Cheese Packaging



Workflow for Chocolate Packaging



Workflow for Dry-fruit Packaging

Fig. 3. Food Packaging Workflows

lookahead is established. As the alternating path of cardboard packaging in the dry-fruit assembly-line is not used for milk-products, the assembly-line is not suitable in this migration situation, in spite of the existence of the polythene packaging option inside the proposed new assembly.

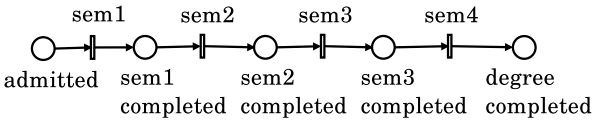
The above example of dynamic workflow migration situation occurs in the context of resource maintenance, which is an assembly line in this case. It can be observed that the necessity of comparing the future of the running cases with the available assembly line is vital to finish the cases by dynamically migrating them. Clearly, history based consistency models do not serve a useful purpose, whereas a lookahead model captures the consistency requirement.

5 Accommodative Lookahead Consistency

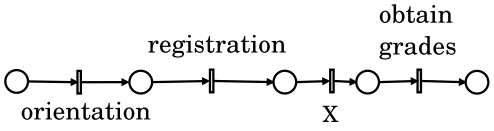
This class of lookahead consistency notion is a weaker one as compared to the earlier class. The accommodative lookahead consistency can be defined to permit new alternatives which are not found in the old net, in addition to the existing traces. In this model, if a trace is possible in the old workflow, it is also possible in the new workflow. However, the converse is not required.

Definition 2 *Following the terms* m, m', m_f, m'_f, F_m, F'_m *as used in Definition 1, Accommodative Lookahead Consistency* $m \ddot{\diamond} m'$ *is defined by the following trace equivalence condition:* $\forall \sigma \in F_m$, $\sigma \in F'_m$. *In other words,* $F_m \subseteq F'_m$.

From the definition we can note that the accommodative lookahead consistency between m and m', i.e. $m \ddot{\diamond} m'$, satisfies the trace equivalence condition (i) of Definition 1. Therefore, strong lookahead consistency between two markings implies accommodative lookahead consistency between them as well, i.e. $m \blacklozenge m' \implies m \ddot{\diamond} m'$. Now we present a dynamic process migration scenario based on accommodative lookahead model.

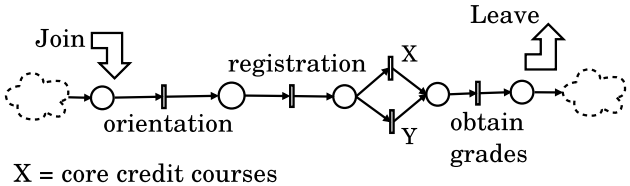


Old and New Workflows for Two-year Masters Program



X = core credit courses for sem1, 2, 3
X = project for sem4

Old Subprocess for Each Semester



X = core credit courses
Y = core credit courses + electives

Foreign University Process

Fig. 4. Student Exchange Program

In an academic curriculum process depicted in Fig. 4, the old process in the migration situation represents a 4-semester masters program. Semesters are sequences of tasks involving orientation, registration, course work and grade reports. Course work comprises of credit courses in the first three semester, and a project in the last semester. The backlog credits are carried over into the next semester. In the process shown in the figure, the semester activities are firstly shown as single transitions of higher level, and they are expanded as a generic subprocess.

A migration situation arises when a student applies for an academic exchange program and joins a foreign university to replace a portion of her academics in the host institute. The student comes back and joins into the old process after completing the exchange credits.

The rules for the exchange program are outlined as follows:

- One student can join an exchange program to study in the foreign university in her second or/and third semester.
- The minimum cut-off CPI for availing the exchange program to join course work in a foreign university is 8.0.
- During the study in the foreign university, a student must complete the courses equivalent to the core credit courses in the home university.
- A student with CPI above 9.0 can join additional honors credit courses as electives.

The foreign subprocess that can replace sem-2 and/or sem-3 activities in the in the old 2-year program to avail the exchange program is highlighted in Fig. 4. It is only a part of a bigger process in the foreign university. A student is allowed to migrate if equivalent core credit courses are available in that semester. The list of courses offered at the foreign university website is to be consulted externally. Therefore, an application for migration are processed at states (markings) *sem-1 completed* or *sem-2 completed*. As the foreign university course work also offers the option of performing elective courses in addition to the core courses, the host institute permits the additional paths.

The above example brings us to the application of accommodative lookahead consistency at the point of migration. This migration scenario shows that a combination of past and lookahead parameters can also be used for deciding migratability. The set of new traces possible for a migrating student is a superset of the old. In this case, several historic or present parameters such as CPI, position in the academic calender are also used to determine the points of migration.

6 Weak Lookahead Consistency

The third kind of lookahead consistency model is the *weak* model. A state of the old workflow is consistent with a state in the new workflow by the weak lookahead model if at least one of the possible future traces is retained in the future of the new net. However, the future of the new net may have additional alternative traces.

Definition 3 *Following the terms* m, m', m_f, m'_f, F_m, F'_m *as used in Definition 1, Weak Lookahead Consistency* $m \diamond m'$ *is defined by the following trace equivalence condition: For non-empty* F_m, $\exists \sigma \in F_m$ *such that* $\sigma \in F'_m$. *For empty* F_m, $F_m = F'_m$.

It can be observed that, quantifier $\forall \sigma \in F_m$ in Definition 2 is enough to find one such case required in Definition 3. Therefore, $m \ddot{\diamond} m' \implies m \diamond m'$. Moreover, we can obtain $m \blacklozenge m' \implies m \diamond m'$ from the already established relation $m \blacklozenge m' \implies m \ddot{\diamond} m'$.

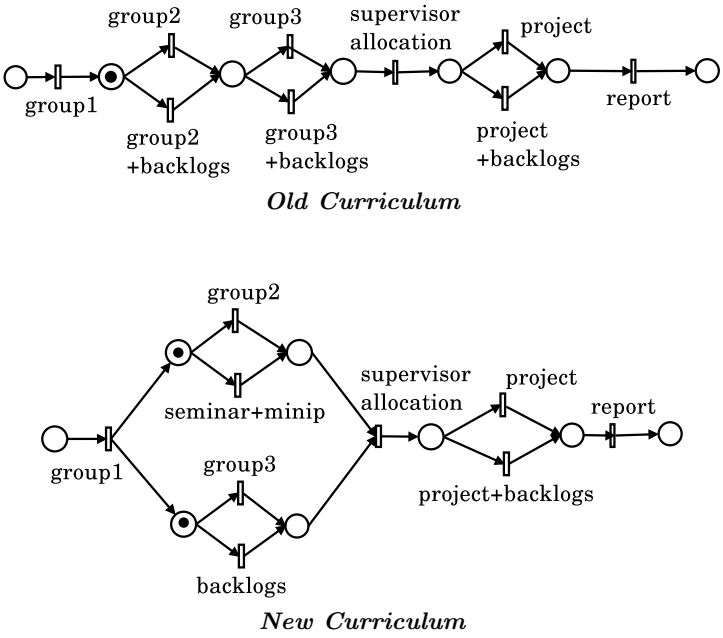


Fig. 5. Academic Curriculum Revision

Fig. 5 depicts an example case of academic curriculum revision of a 4-semester masters program. In the old curriculum, credits for three groups of courses are to be completed in the first three semesters. Alternate branches with backlog credits are available. The allocation of a project-supervisor is arranged before the final semester project work. The program ends with project reporting. The revised curriculum emphasizes on the depth in course work rather than the extent of the covered syllabus, by reducing the number of compulsory credits for the under-performing students. After completing group-1 courses in the first semester, the students have to perform two groups of courses. They can take up group-2 and group-3 courses as per the old curriculum, or they have the option of a seminar and a mini-project course. The students having backlogs can not register for group-3 courses.

A student in the old curriculum can migrate from current state into a state in the new curriculum, the design makes available at least one path in the new which is exactly a path in the old. However, new alternatives are also open to the migrating student. One set of equivalent markings by this criteria is shown in the figure. It can be noted that for the single token in the old workflow, two tokens are generated in the new workflow, which together constitute the consistent mapping. The new workflow preserves the traces *group2, group3, supervisor allocation, project, report* and *group2, group3, supervisor allocation, project+backlog, report* from the old workflow and not the others. Some traces of the old workflow are suppressed in the new workflow. In this way, the weak lookahead model can be applied to define flexible but at least minimally compatible workflows.

7 Instance Adaptations through Lookahead Consistency

In the previous sections, we have developed the lookahead consistency models and illustrated their applicability in several dynamic workflow migration situations. In this section, we first discuss the computation of weak lookahead consistent markings in the new net. Further, we discuss the approach of enforcing lookahead consistent execution on migrating instances in situations in which a weaker model is available in the new net. The *accept/reject branching* algorithm is then presented to support the lookahead consistency enforcement approach. The workflow nets considered in the following algorithms are considered to be acyclic. Moreover, all transition labels in the nets are assumed to be unique.

7.1 Algorithms for Lookahead Consistency

Algorithm 1 computes the weak lookahead consistent markings in the new net by replaying the lookahead traces possible in the old net. As only acyclic nets are considered, the set $Traces$ of lookahead traces is finite. For a set of computed traces in the old net, all of them may not replay in the new net. The algorithm finds out the markings in the new net each of which can replay at least one of the traces. The outputs of the algorithm are set S of weak lookahead markings and set L containing the preserved lookahead traces in the new net.

Algorithm 1: Computation of Weak Lookahead Consistent Marking

Input: Old WF-net $N = (P, T, F)$, Marking M in N, New WF-net $N' = (P', T', F')$
Result: Set of Markings S in N', Set of Preserved Lookahead Traces L

Let M_f and M'_f are the terminal markings in N and N' respectively
$Traces \leftarrow \{\sigma \mid M \xrightarrow{\sigma} M_f\}$
if $Traces = \{\}$ **then**
    $S \leftarrow M'_f$
    $L \leftarrow \{\}$
    return
$Traces^r \leftarrow \{\sigma^r \mid \sigma^r$ is reverse of $\sigma, \sigma \in Traces\}$
$F'_{edit} \leftarrow \{(x, y) \mid (y, x) \in F'\}$
$N'_{edit} \leftarrow (P', T', F'_{edit})$
$S \leftarrow \{\}$, $L^r \leftarrow \{\}$
while $Traces^r \neq \{\}$ **do**
    Let σ^r be a member of $Traces^r$
    if $M'_f \xrightarrow{\sigma^r} M'_e$ *in* N'_{edit} **then**
        $S \leftarrow S \cup M'_e$
        $L^r \leftarrow L^r \cup \sigma^r$
    $Traces^r \leftarrow Traces^r - \{\sigma^r\}$
$L \leftarrow \{\sigma \mid \sigma$ is reverse of $\sigma^r, \sigma^r \in L^r\}$

First it computes $Traces$, the set of lookahead traces in the old net. If set $Traces$ is empty, it indicates that the old net is already terminally marked. Therefore, the only lookahead consistent marking in the new net is the terminal marking M'_f. Also, since there is no lookahead trace in the old instance, the set of preserved lookahead traces in the new net is also empty. Therefore, the algorithm terminates here after computing these boundary outputs. Lines 3-6 handle this boundary case.

For non-empty $Traces$, it flips the member traces and stores them into set $Traces^r$. Lines 8-9 reverse the arcs directions in the new net N' and stores the reversed net as N'_{edit}. For each of the traces in $Traces^r$, the algorithm looks for its occurrence in N'_{edit}. Finally, set L contains those original lookahead traces that can be replayed in N', and set S contains the weak lookahead consistent markings in N'.

The algorithm thus looks for all traces starting from the current marking in the old net, collecting all new markings corresponding to these traces in set S. Since, this collection is a set, a marking appears only once even though it can trigger more than one lookahead traces due to fork-join patterns. The set S can however have multiple markings in certain configurations as given in Fig. 6. For the example given in this figure, the algorithm starts with $Traces = \{t_1t_3, t_2t_3\}$ and ends with computing $S = \{\{p'_1\}, \{p'_2\}\}$, $L = \{t_1t_3, t_2t_3\}$.

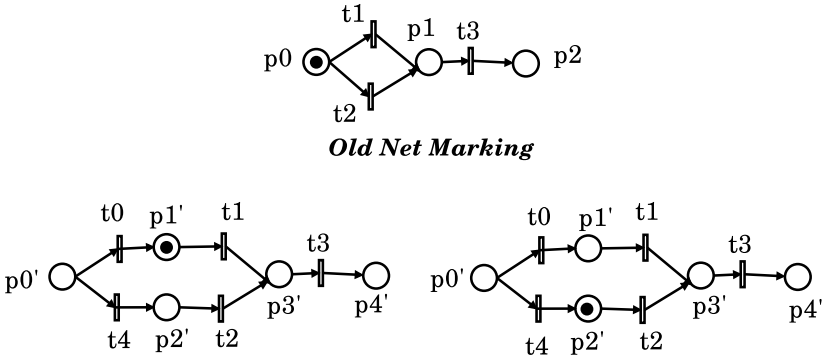


Fig. 6. An Example Case for Algorithm 1

7.2 Support vs. Enforcement of Lookahead Consistency

The motivation behind enforcing lookahead consistency is as follows. Given a marked old net and a new net schema, even though the lookahead consistent marking is supported in the new net, due to the generality of the new net, the

tokens can not be guaranteed to follow the execution path as prescribed by the old model. For example, consider two consistent markings are shown in the milk-product packaging and dry-fruit packaging workflows of Fig. 3 as per the weak or accommodative consistency model. As noted out earlier, since they do not satisfy the strong model, the assembly line of dry-fruit packaging can not take over the task of milk-product packaging. However, if we disable the traces in the new workflow which violate strong lookahead, the same can still be ensured for conforming to the desired post-migration paths. As a result, this approach achieves stricter consistency as intended for the migrating instance, which is not otherwise enforced by the new schema.

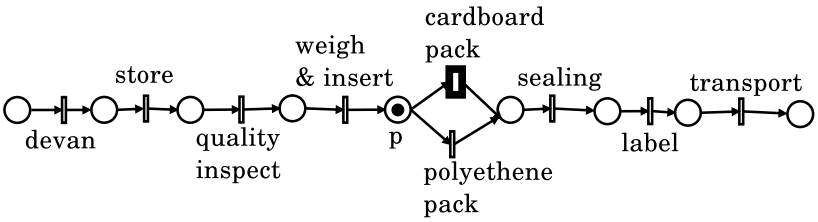


Fig. 7. Output of Accept/Reject Branching Algorithm

For the marking given in the milk-product packaging workflow in Fig. 3, Fig. 7 depicts the consistent marking and transition to be blocked to enforce strong lookahead consistency for the instance migrating into the dry-fruit packaging workflow. The only transition to be blocked in this case is shown as a box with thick border.

Next, the *accept/reject branching algorithm* given in Algorithm 2 identifies such transitions that need to be blocked to enforce the consistency preserving lookahead executions. It is assumed that a suitable implementation mechanism for disabling the transitions is available in the workflow management system.

Algorithm 2: Accept/Reject Branching

Input: WF-Net $N = (P, T, F)$, Marking M in N, Set of lookahead traces Σ
Result: Set of transitions T_{block}

1 let $path(e_0, e_0) =$ `TRUE`
2 let $path(e_i, e_j)$ be a boolean function indicating the existence of directed path from net element (place or transition) e_i to net element e_j
3 $P_{exchoice} \leftarrow \{\ p \mid (p, t_i), (p, t_j) \in F\ , i \neq j,\ \exists p_0$ such that $M(p_0) = 1, (p_0, p) =$ `TRUE` $\}$
4 $T_{potential} \leftarrow \{\ t \mid t \in T, p \in P_{exchoice}, (p, t) \in F\ \}$
5 $T_{lookahead} \leftarrow \{\ t \mid \sigma \in \Sigma, t \in T_{potential}, t \in \sigma\ \}$
6 $T_{block} \leftarrow T_{potential} - T_{lookahead}$

The inputs to the algorithm is a marking in the new net and the set of desired lookahead traces Σ which is set to L, the output of Algorithm 1. As output it produces the set of transitions T_{block} which can be blocked to disable all the lookahead traces that are not in Σ.

First the algorithm identifies the *choice gateways* which are not yet traversed by the tokens in the new net. In a sequence, disabling the head-task prevents the whole sequence. Paths following exclusive-choice gateways are sequences with head-tasks as the first tasks after the choice. The algorithm finds out the set of tasks $T_{potential}$ which holds all the head-tasks following the choice gateways in the new net. Discarding those tasks from $T_{potential}$ which are not in the lookahead traces gives the remainder portion of the net which should be left as active. In this way, the algorithm finds out the tasks to be blocked as the set T_{block}, which is $T_{potential} - T_{lookahead}$.

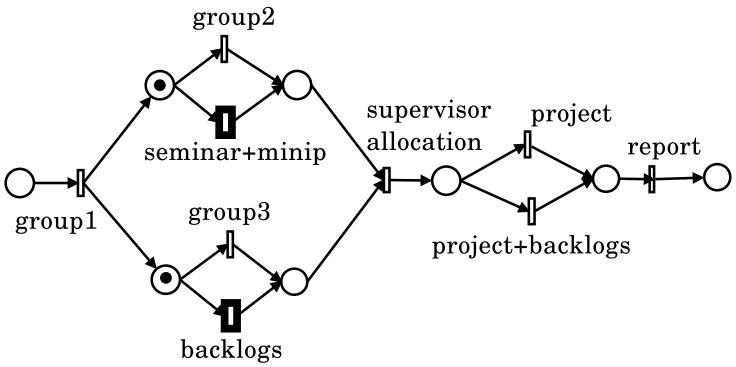


Fig. 8. Blocked transitions in the new workflow shown in Fig. 5

It can be noted that blocking of lookahead transitions for the migrated instances may not be an appropriate solution in a given application. For instance, consider the case of migrated instance in Fig. 8 of the academic workflow process shown in Fig. 5. The blocked transitions are shown as boxes with thick border. As a result of blocking these transitions, the migrating student can proceed only through *group2* and *group3* courses. However, due to the additional constraint of the university that a student having backlogs can not register for *group3* courses, such students can not be migrated since there is no path left for them in the new net. Also, migrating students can not take seminar and mini-project courses available in the changed curriculum. Therefore, with reason of retaining flexibility, blocking of transitions is not suitable for this situation. On the contrary, the approach of blocking transitions is suitable for those cases where migration is inevitable, and yet sticking to the old execution paths is necessary for application semantics. A case study of a library resource acquisition workflow given in Section 7.3 describes one such suitable migration scenario for accept/reject branching.

The following conditions summarize the inferences regarding the class of lookahead consistency that can be drawn from the outputs of the two algorithms.

1. $L \neq \{\}$ confirms weak lookahead consistency. (Algo. 1)
2. In addition, $|S| = 1$ and $L = Traces$ confirms accommodative lookahead consistency. If S contains more than one markings, no single marking can fire all lookahead traces since there are no duplicate transitions. Hence, the condition $|S| = 1$. (Algo. 1)
3. In addition, $T_{block} = \{\}$ confirms strong lookahead consistency. (Algos. 1, 2)
4. $S = \{\}$ implies absence of lookahead consistency.
5. If $T_{block} \neq \{\}$, blocking of the transitions in set T_{block} is a mechanism to enforce the desired lookahead traces. This converts accommodative lookahead to strong lookahead. In the case of weak lookahead, the lookahead traces found can indeed be enforced by blocking these transitions.

7.3 Case Study of a Library Resource Acquisition Workflow

A library resource acquisition workflow case is considered for this case study. Processes in this system are orders of the kind *firm orders*. (There are other types of library acquisitions such as serial subscriptions, standing orders and blanket orders which are not considered in the case study.) In the old system, the institute has separate processes of resource acquisition for the academic departments and for the central library as shown in Fig. 9a and Fig. 9b respectively. It is proposed to merge these processes into a single one. We first describe the old processes after which the new process is outlined. After this, a consistent lookahead based migration solution is worked out.

The Old Departmental Process The departmental process can be followed only for acquisition of hard-copies. First the bibliographer has to prepare the list of books to be purchased. The overall expense is then estimated and an application is sent next to the department office for approval of the budget. Arrival of the funding approval initiates the negotiation procedure with the vendors. In the case of rejection of the funding application from the department office, the workflow can proceed in one of the two ways. If the applicant can arrange money from her/his project funds, the workflow can proceed to join the flow of usual acquisition process by initiating the price negotiation. Otherwise, in the case of unavailability of funding, the acquisition case is dropped. Next, the payment is carried out for the agreed price as a confirmation of the purchase order to the vendor. Delivery of the books along with the invoice completes the resource acquisition. The workflow finishes after recording the acquisition details in the department resource database and with cataloging of the acquired resources.

The Old Central Library Process The central library workflow follows similar logic for hard-copy resource purchase, though the funding agencies are different. All acquisition requests are sent to the academic office for funding approval.

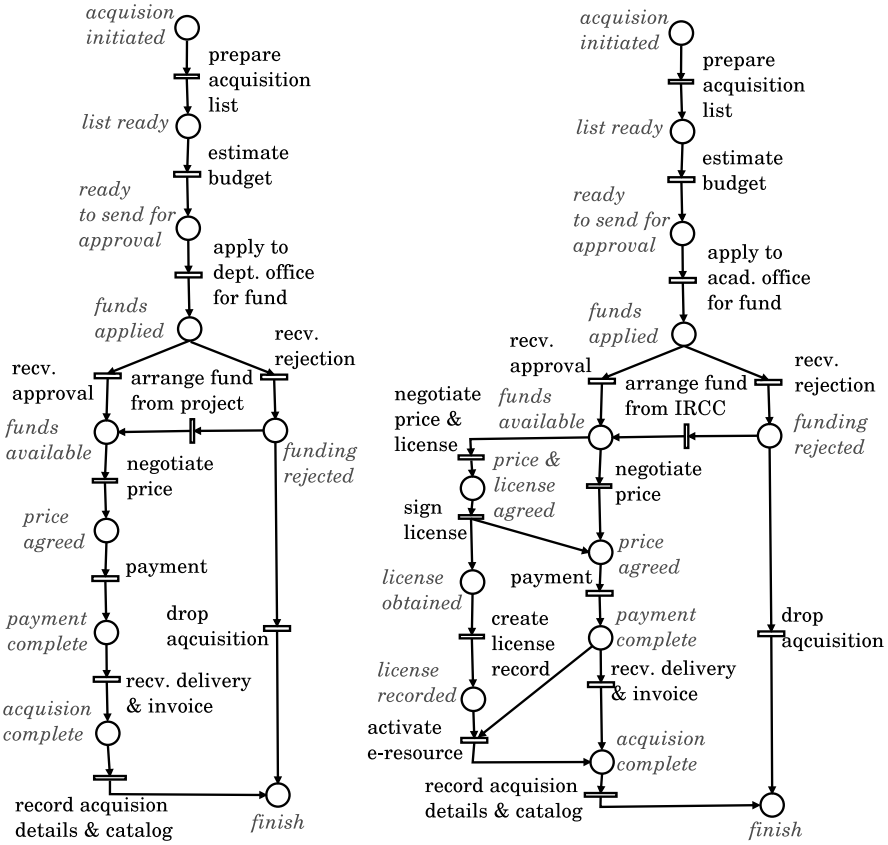


Fig. 9. Library Workflows for Resource Acquisition

In the case of rejection, either the Industrial Research and Consultancy Center (IRCC) supports the funding issues, or the case has to be dropped. In addition to the above, the central library workflow supports purchases of e-resource. In the case of e-resource acquisition, additional activities for license negotiation and agreement are incorporated in the workflow. Once the license is signed, its copy is received by the library which is recorded in the central library database. After payment, the e-resource is activated. According to the license agreement, this step can involve storing a local copy of the resource or enabling password protected access of the document residing on its remote host. Cataloging of the resource is performed on the central library database, which wraps up the process.

Process Re-engineering A process re-engineering team decides to merge all the departmental workflows with the central library workflow due to the following reasons.

- A consolidated storage of the library resources across all the academic departments can achieve better resource sharing among several departments, to the benefit of multi-disciplinary studies and projects.
- E-resource acquisition can be performed using the departmental or individual project funds introducing a level of flexibility.
- Individual departments can leave the responsibilities of library-staff recruitment and related activities to the central library authority reducing the redundant efforts in the old system.

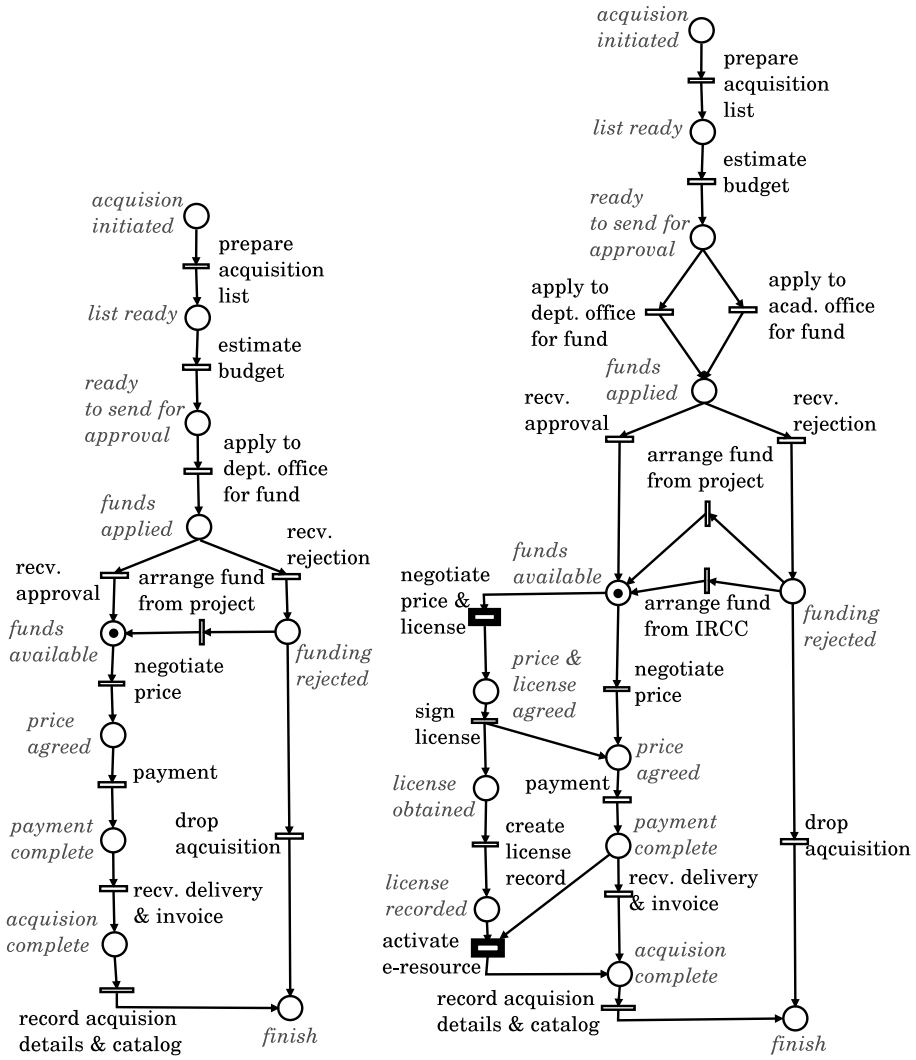
As a result, the department resource databases has to be merged with the central library database. Secondly, the running instances need to be migrated in a consistent way by applying appropriate models as per the needs of individual old cases. It is observed that the task *record acquisition details & catalog* in the old process updates department resource database, whereas, in the new merged process it updates the central library database. This requires migration of all old incomplete instances into the new schema.

Lookahead Based Consistent Dynamic Instance Migration Fig. 10a shows such an on-going workflow instance that is in state *funds available*. The migration scenario requires weak lookahead consistency criterion for the safe migration of the running instances in order to complete the hard-copy acquisition process from the department. The merged workflow schema and the migrated marking is shown in Fig. 10b.

In addition to the migration, the situation requires that the same path be enforced for the migrating instance without giving the additional flexibility of the alternative of e-resource purchase. Therefore, the execution of these running cases must be prevented from traversing the path for e-resource purchase, which is achieved by blocking the transitions which are output of the accept/reject branching algorithm. The traces of both the algorithms are given below.

- $Traces$ = { *negotiate price*, *payment*, *recv. delivery & invoice*, *record acquisition details & catalog* }.
- L = { *negotiate price*, *payment*, *recv. delivery & invoice*, *record acquisition details & catalog* }.
- S = { *funds available* }.
- $P_{exchoice}$ = { *funds available*, *payment complete* }.
- $T_{potential}$ = { *negotiate price*, *negotiate price and license*, *recv. delivery & invoice*, *activate e-resource* }.
- $T_{lookahead} = L$.
- As a result, T_{block} = { *negotiate price and license*, *activate e-resource* }.
- It can be noted that, $L \neq \{\}$ ensures weak lookahead consistency. Further, $|S| = 1$ and $L = Traces$ ensures accommodative lookahead consistency. However, $T_{block} \neq \{\}$ implies that strong consistency is not supported by the modified workflow schema.

The transitions in set T_{block} are shown as boxes with thick border in Fig. 10b. These transitions can be blocked in the new workflow execution environment



(a) ***Old Instance in-progress*** (b) ***Migrated Instance in Merged Workflow***

Fig. 10. Instance Migration into the Re-engineered Workflow

only for this migrating instance to enforce its required lookahead consistency as discussed above.

8 Discussion and Conclusion

In the literature, execution history based consistency notions are widely adopted wherever dynamic evolution and adaptation of workflows are considered. The proposed new model of lookahead consistency uses the remainder of the workflow instead of the completed history in defining consistency between states in

a migration scenario. Three broad classes of the lookahead consistency were brought out with illustrative examples. The weak lookahead model looks for preservation of at least one lookahead trace of the old instance across the new net. For accommodative lookahead model, preservation of all lookahead traces of the old instance is necessary. The strong model requires the lookahead traces of the old instance and its migrated marking in the new net to be same. Strong lookahead implies accommodative lookahead, which in turn implies weak lookahead consistency.

Besides their usefulness and the inter-relationships among the models, two related algorithms were also presented for deciding lookahead consistency, computing token transfer, and for lookahead trace enforcement. The new set of lookahead models provide future-centric approaches of varying flexibility for token transfer in dynamic migration scenarios with applications to process re-engineering and maintenance. A case study of library resource acquisition workflow demonstrated the relevance of the approach.

As the past execution traces are not taken into account in the context of lookahead based consistency models, these can not be used stand-alone in situations requiring history equivalence. However, the lookahead models can also be applied in combination with history equivalence as per the needs of business goals. The algorithms for acyclic nets were implemented in GNU Octave [20]. We are working on an integration of this implementation with the facility of blocking the unintended transitions in a workflow engine [21] environment, and an extension of the ideas to work with cyclic nets.

References

1. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: Proceedings of conference on Organizational computing systems, ACM (1995) 10–21
2. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data & Knowledge Engineering **24**(3) (1998) 211–238
3. Sadiq, S.W.: Workflows in dynamic environments - can they be managed? In: In Proceedings of the 2nd International Symposium on Cooperative Database Systems for Advanced Applications. (1999) 27–28
4. Reichert, M., Dadam, P.: Adeptflex—supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems **10**(2) (1998) 93–129
5. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on. (Jan 2001) 10 pp.–
6. Dias, P., Vieira, P., Rito-Silva, A.: Dynamic evolution in workflow management systems. In: Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on, IEEE (2003) 254–260
7. Rinderle-Ma, S., Reichert, M., Weber, B.: Relaxed compliance notions in adaptive process management systems. In: Conceptual Modeling-ER 2008. Springer (2008) 232–247

8. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. In: Transactions on Petri Nets and Other Models of Concurrency II. Springer (2009) 115–135
9. Sun, P., Jiang, C.: Analysis of workflow dynamic changes based on petri net. Information and Software Technology **51**(2) (2009) 284 – 292
10. Pradhan, A., Joshi, R.K.: Token transportation in petri net models of workflow patterns. In: Proceedings of the 7th India Software Engineering Conference. ISEC '14, ACM (2014) 17:1–17:6
11. van der Aalst, W.M.: Exterminating the dynamic change bug: A concrete approach to support workflow change. Information Systems Frontiers **3**(3) (2001) 297–317
12. Cicirelli, F., Furfaro, A., Nigro, L.: A service-based architecture for dynamically reconfigurable workflows. Journal of Systems and Software **83**(7) (2010) 1148 – 1164
13. van der Aalst, W.M.: The application of petri nets to workflow management. Journal of circuits, systems, and computers **8**(01) (1998) 21–66
14. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems–a survey. Data & Knowledge Engineering **50**(1) (2004) 9–34
15. van der Aalst, W.M., Basten, T.: Inheritance of workflows: an approach to tackling problems related to change. Theoretical Computer Science **270**(1) (2002) 125–203
16. Milner, R.: Communicating and mobile systems: the pi calculus. Cambridge university press (1999)
17. Ryu, S.H., Casati, F., Skogsrud, H., Benatallah, B., Saint-Paul, R.: Supporting the dynamic evolution of web service protocols in service-oriented architectures. ACM Transactions on the Web (TWEB) **2**(2) (2008) 13
18. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
19. Reichert, M., Weber, B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer Science & Business Media (2012)
20. Eaton, J.W., Bateman, D., Hauberg, S.: GNU Octave version 3.0.1 manual: a high-level interactive language for numerical computations. CreateSpace Independent Publishing Platform (2009) ISBN 1441413006.
21. Pradhan, A., Joshi, R.K.: Architecture of a light-weight non-threaded event oriented workflow engine. In: The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014. (2014) 342–345