Customized Atomicity Specification for Transactional Workflows

Wijnand Derks
KPN Research, Netherlands
w.l.a.derks@kpn.com

Juliane Dehnert*
Technische Universität Berlin
dehnert@cs.tu-berlin.de

Paul Grefen
Center for Telematics and Information Technology (CTIT), University of Twente, Netherlands
grefen@cs.utwente.nl

Willem Jonker
KPN Research, Netherlands
willem.jonker@kpn.com

Abstract

This paper introduces a new approach for specifying transaction management requirements for workflow applications. We propose independent models for the specification of workflow and transaction properties. Although we distinguish multiple transaction properties in our approach, we focus on atomicity in this paper. We propose an intuitive notation to specify atomicity and provide generic rules to integrate the workflow specification and the atomicity specification into one single model based on Petri Nets. The integrated model can be checked for correctness. We call this correctness criterion relaxed soundness as a weaker notion of the existing soundness criterion. We can relax the correctness criterion because we rely on run-time transaction management. A real life example shows the applicability of the concepts.

1. Introduction

To improve the quality and efficiency of service provisioning, many enterprises have developed workflow applications to automate their processes. As services become increasingly complex, so become these processes and the requirements for reliability of these applications. Transaction management provides a means to ensure correctness in the presence of concurrency and failures. Therefore transaction models can be applied to workflow applications to achieve increased reliability.

Present approaches try to map existing transaction models to workflow specifications or introduce new advanced transaction models to fit the requirements of complex workflow applications. However, transaction requirements for workflows are often application dependent and therefore we require a model to specify transactional properties independently from existing models.

The approach we take is to specify the workflow and transaction requirements independently. After specification, we integrate the workflow process and the transaction requirement specification to be checked for consistency. We consider the specification consistent, if the workflow process can be executed according to the transactional requirements. In this paper, we restrict ourselves to the transactional property *atomicity*. Atomicity constraints define an existence relation between tasks in a group, e.g. that either all tasks in the group should execute, or no task of the group should execute.

To be able to integrate the specification of workflow and atomicity, we model both with Petri Nets. For the workflow specification we adopt Workflow Nets, which is based on Petri Nets and for the specification of the atomicity requirements, we provide rules for the translation of the atomicity requirements into Petri Nets. Hence both models can be integrated. The consistency of the integrated model is then determined by the *relaxed soundness* criterion. Relaxed soundness is a relaxation of the soundness criterion introduced in [1]. We can relax soundness, because we rely on run-time transaction management to allow more freedom in the specification of the workflow. We show the applicability of the proposed approach using an example from a cross-organizational setting.

2. Related work

In the nineties workflow applications were identified as an important application domain for transaction management. In [10] Sheth and Rusinkiewicz introduce *transactional workflows* as workflows with transaction support. Since then, a lot of work has been done to integrate workflows and transaction models. Reuter [9] introduces a transactional workflow specification and execution framework called ConTracts. It supports execution of (a group of) tasks that preserve strict ACID

0-7695-1128-7/01 $10.00 © 2001 IEEE

properties. However, atomicity cannot be specified as an independent transaction property.

Similar to the ConTracts approach, in [5] inter-task dependencies describe transactional dependencies between atomic tasks. The focus of this work is on enforceability of the inter-task dependencies. However, atomicity specification remains implicit in the specification.

The work of Adam et al. [4] builds further on the approach of [5] and offers a formal framework for specification of transactional workflow applications based on Petri Nets. However, similar to [5] transactional properties between activities are expressed implicitly by operational inter-task dependencies, whereas we specify them explicitly by separating them from the workflow specification.

Other approaches apply advanced transaction models to workflows. Examples are the Exotica project [2] where the SAGA model and the Flexible Transaction models are integrated within a Flowmark process specification. This way, atomicity remains implicit in the workflow specification. In WIDE [6] the extended SAGA model and the nested transaction model were incorporated. WIDE defines safe-points to identify process states that separate semantic units of work. This is similar to atomic units, although safe-points are not customizable independently from the SAGA model. Recently, in the ESPRIT project CrossFlow the WIDE approach was extended to cross-enterprise workflows [12].

Similar to our approach Leymann et al. [7] specify transaction properties and the workflow independently. by customizable *atomic spheres* and *spheres of joint compensation*. However, they approach atomicity from a persistence perspective, whereas we define atomicity as a property of the workflow. Persistence can be specified in addition to our atomicity specification. Similarly, Alonso [3] expresses the need for separate transactional semantics to be applied to groups of activities. He distinguishes spheres of atomicity, isolation and persistence. The sphere of atomicity is similar to Leymann's atomic sphere. However, in [3] atomicity and recovery are coupled, whereas we consider atomicity separately from recovery.

3. Workflow Process

For the specification of a workflow process definition we use Petri Nets, because Petri Nets have a clear and precise definition and Petri Nets allow us to make use of many existing analysis techniques and tools [4]. Van der Aalst [1] applies Petri Net theory to workflow specification and introduces Workflow Nets (WF-Net). In a WF-Net a task is modeled by a transition and intermediate states are modeled via places. A token in the source place *i* corresponds to a case which needs to be handled, a token in the sink place *o* corresponds to a case that has been handled. The process state is defined by a marking. A marking *M* is the distribution of tokens over places. In this paper we take the definitions of a Petri Net and WF-Net from [1]:

Definition (Petri Net). A Petri Net is a triple (P,T,F):

- P is a finite set of places,
- T is a finite set of transitions ($P \cap T = \varnothing$),
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation).

Definition (WF-Net). A Petri Net PN = (P,T,F) is a WF-net (WorkFlow Net) if and only if:

(i) PN has two special places: *i* and *o*. Place *i* is a source place and place *o* is a sink place.

(ii) If we add a transition *t** to PN which connects place *o* with *i*, then the resulting Petri Net is strongly connected.

A Petri Net is *strongly connected* if and only if for every pair of nodes (i.e. places and transitions) *x* and *y*, there is a path leading from *x* to *y*.

Figure 1 shows a WF-Net. The example represents a cross-organizational workflow process that involves two companies C1 and C2. Both companies co-operate, but have independently specified processes. Company C1 takes an order from a customer and outsources the delivery of the order to company C2. Although company C2 delivers the product, company C1 takes responsibility for the billing of the order. Therefore company C1 checks whether the product should be delivered to the customer by the *check_credit* task. The *check_credit* task results in either a *not_ok* or *ok*. We model these outcomes in the picture explicitly as tasks for clarification purposes. The processes are integrated only at the beginning and end of both processes. The task *start_co-op* indicates the start of the co-operation of the specific case and task *end_co-op* ends the co-operation. Note that this way, the process specification of C1 and C2 needs no adjustments, and therefore co-operation is easily established. This is especially important in dynamic cross-organizational outsourcing, where co-operation relationships are of short duration.

While the preparation of the order is processed at company C2, company C1 checks whether the customer has sufficient credits to pay the order. If this appears not the case, company C2 should not deliver, but cancel the delivery. Note that this dependency between both sub-processes is not specified in Figure 1. In particular, the *cancel_order* at C2 should only be executed if and only if the result of the *check* on the C1 side was *not ok*.

This missing constraint is purely *functional*, which means that it does not say anything about ordering but only expresses an existence relationship. One way to

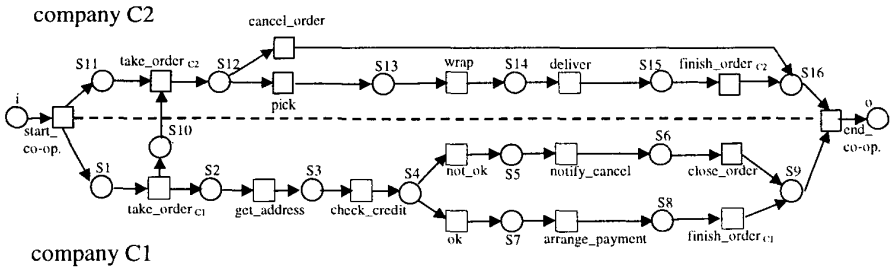

Figure 1. WF-Net order processing

implement this constraint, both processes could be synchronized at the decision points. For example, two places and corresponding arcs could be added to synchronize transition *ok* and *pick*, and transition *not_ok* and *cancel_order*. The process specification would then serialize the execution: first execute the customer check and then execute the ordering. This implementation is pessimistic, because it avoids inconsistent execution beforehand.

However, in case the delivery process takes long and the customer check takes long, this pessimistic scheduling would be inefficient. This is especially undesirable if it is very rare that a customer check results in a *not_ok*. A more efficient way of implementing the atomicity requirement would then be to just start the delivery of the order to the customer hoping the customer check will be OK, i.e. following an *optimistic* approach. Only in the rare case that the decision *not_ok* was taken, the order should be returned to stock, i.e. go back to S12 and then cancel the order after all. Recovery from this inconsistent state could then be supported by transaction management, e.g. compensation [12]. Note that we do not model compensation explicitly, but assume each task to have an inverse task that undoes all of its effects.

Because different approaches could be followed to ensure the atomicity requirement, we consider this synchronization of processes an implementation issue. Therefore we want to abstract from synchronization and express an execution dependency between tasks independently from the ordering. For this we introduce *atomicity spheres.*

4. Atomicity

We define atomicity in terms of completeness, which means that an atomic unit of work is either executed completely or not at all. The smallest granularity of atomicity is the task and we consider a single task to be atomic. We extend the notion of atomicity to a set of tasks and consider these tasks to be contained in an *atomicity sphere*. We consider an atomicity sphere to *execute* if at least one task in the sphere executes.

Strict-atomicity defines the basic atomic behavior, i.e. all tasks in a *strict atomicity sphere* have to be executed or no task in the sphere at all. This strict atomicity constraint can be relaxed into two dimensions: alternatives and exceptions.

Alternative-atomicity specifies exclusive execution of *combinations of atomicity spheres*. This notion defines that the execution of tasks is according to the defined atomicity spheres and requires that *at most one* specified *combination of atomicity spheres* executes. *Alternative atomicity* is introduced for atomicity specification in case alternative ways of execution are allowed. Considering the example in Figure 1 it would be desirable to specify that if the task *take_orderC1* is executed, either the task *cancel_order* or the task *finish_orderC1* is executed as well. Note that it is not possible to specify the alternative execution as two strict atomicity spheres. This would force the execution of both the tasks *cancel_order* and *finish_orderC1*.

Exception-atomicity allows a group of tasks to violate the atomicity constraint in a controlled manner: tasks in an exception-atomicity sphere do not all have to execute, but in case one or more tasks in the sphere do not execute, an exception task is executed. Weakening atomicity is useful in the workflow context to allow for exceptions like timeouts or to allow for atomic units that cannot be forced to behave atomically. Note that an exception-atomicity sphere always introduces an additional task in the workflow specification: the exception task. Therefore the workflow process designer has to incorporate this exception task in the workflow process explicitly.

The notations of the different atomicity constraints are depicted in Figure 2 for examples with six tasks. An atomicity sphere is drawn as a solid box containing the corresponding tasks. Alternatives are denoted by numbered dark bullets that connect atomicity spheres. Exceptions are visualized by a gray exception task. The atomicity requirement in the combination s/n is satisfied if and only if the tasks 1 through 6 either all execute or none of them. The atomicity sphere in the combination s/e is satisfied if all tasks execute or no task executes while the exception task does not execute, or, if *some* tasks together

with the exception task execute. The atomicity requirement as shown in the cell a/n is satisfied, if the tasks 1 through 4 all execute and tasks 5 and 6 do not execute, or the tasks 3 through 6 all execute and neither task 1 nor 2. Another valid option is that no task executes. The atomicity requirement in the combination a/e is satisfied similar to a/n, but in addition allows violation of a combination if and only if the exception task is executed. Note that even in the case of a violation of the alternative, spheres {1,2}, {3,4} and {5,6} each still behave as a strict atomic unit, i.e. either all task execute or none. Execution of all tasks 1 through 6 including the exception task is also valid.

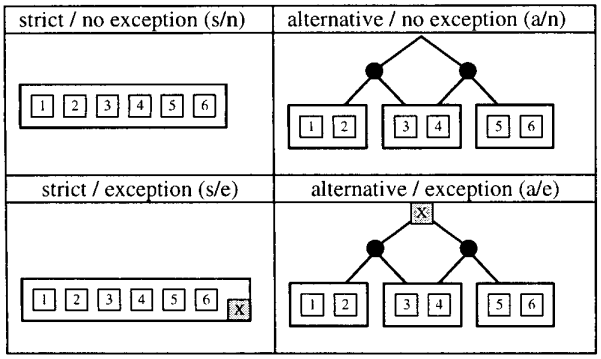

Figure 2. Notations for the different atomicity types

4.1. Application in the example

To demonstrate the use of atomicity spheres, we have identified atomicity requirements in the example of Figure 1. The tasks *pick, wrap* and *deliver* are contained in a *strict atomicity sphere*. This means that those three tasks cannot execute independently. The intuition is that a package that is picked should always be packaged and delivered. Also the task *take_order_C1* and *notify_cancel* are each contained in a strict atomicity sphere for the sake of consistent notation. Note that this corresponds to the normal task execution semantics, because we consider a task to be atomic by itself.

We specify *alternative atomicity* for tasks {*take_orderC1*}, {*pick, wrap, deliver*} and {*notify_cancel*}. The alternatives are specified such that either {*take_orderC1, notify_cancel* } executes, or {*take_orderC1, pick, wrap, deliver* } executes. This implies that the *notify_cancel* task can never execute together with one of *pick, wrap* or *deliver*. This ensures that if the order is taken by C1, either the order is cancelled or the parcel is processed.

Furthermore, we specify an *exception atomicity sphere* containing the tasks *get_address* and *check_*credit to allow partial execution of the customer check. If either of the tasks does not execute, the exception task is executed. In Section 6.1 we show how task *X* is integrated in the process specification.

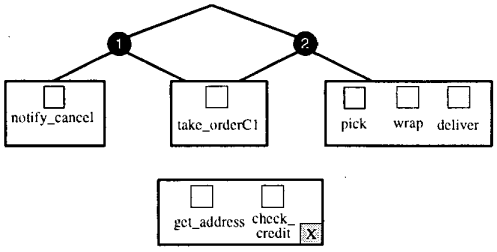

Figure 3. Atomicity specifications in the example

5. Specifying atomicity with Petri Nets

An atomicity-sphere containing a group of tasks specifies the behavior that the tasks should either execute all or none. Figure 4 gives the translation of the strict atomicity sphere with two tasks *1* and *2* into a Petri Net. The transitions *not_at_all* and *all* represent the valid alternative executions of the atomic sphere. The sphere is initialized by a token in *local i* and define it finished with a token *only* in *local o*. Note that this definition implies that the workflow is required to be acyclic and that *not_at_all* must not fire if task *1* or *2* still may be executed. We have incorporated this requirement in our correctness criterion *relaxed soundness*, which is introduced in Section 6.3.

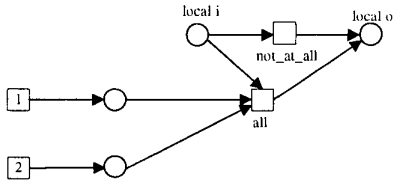

Figure 4. Petri Net pattern for a strict atomicity sphere

5.1. Alternative atomicity

In the case of alternative-atomicity, combinations of atomicity spheres are defined. Alternative-atomicity requires, that *at most one* of those combinations of atomicity spheres be executed. In Figure 2, three atomicity spheres are defined: {1,2}, {3,4} and {5,6}, which we will denote S1, S2 and S3 here. The allowed alternative executions include A1: {1,2,3,4} and not {5,6} (i.e. S1 and S2 and not S3), and A2: not {1,2} and {3,4,5,6} (i.e. not S1 and S2 and S3).

The Petri Net pattern for the alternative atomicity sphere is similar to the strict atomicity sphere and is shown in Figure 5. We have not shown the Petri Net patterns for each of three spheres in the figure for reasons of clarity.

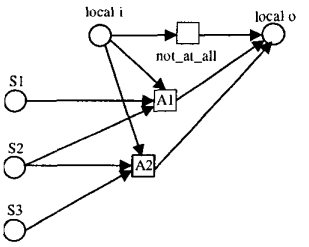

Figure 5. Petri Net pattern for alternative atomicity

5.2. Exception-Atomicity

Figure 6 gives the translation for the example in the s/e-combination (see Figure 2) into a Petri Net pattern for two tasks. The Petri Net incorporates all possible alternative executions of the tasks, i.e. no execution, only task 1 executes, only task 2, or both task 1 and task 2 execute. The execution of only task 1 or only task 2 violates the strict atomicity requirement and should only occur if and only if the exception task is executed. The exception is modeled by the *X* transition.

Note that this way of modeling is exponential, because the powerset of the set of tasks must be present as an alternative. This can be ignored, if the number of tasks contained in an atomicity sphere is small. For the case of larger number of tasks, we have developed a Petri Net pattern that scales linearly with the number of tasks in the sphere. This pattern is omitted for brevity.

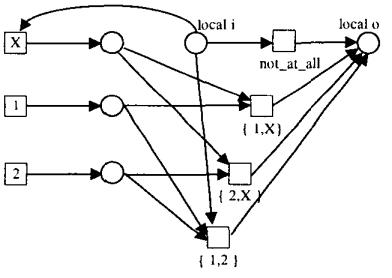

Figure 6. Petri Net pattern for an exception atomicity sphere

The Petri Net pattern of the exception alternative (e/a) combination is similar to the pattern for the exception atomicity sphere. We omit the pattern due to space limitations.

5.3. General construction rules

In the previous paragraphs we have shown the Petri Net patterns for each atomicity type independently. Different atomicity types can also be combined to define complex atomicity constraints. An example is presented in Figure 7.

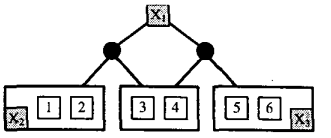

Figure 7. Example of a complex atomicity constraint

The general approach for constructing Petri Nets for complex atomicity spheres is to:

1. construct Petri Net patterns for the strict and exception atomicity spheres,
2. construct Petri Net patterns for the alternative atomicity specification,
3. connect all execution alternatives of the atomicity spheres, i.e. transitions A_x, with the sphere places of the alternative atomicity Petri Net, i.e. places S_x.

6. Model integration

To show that the specification of atomicity spheres is consistent with a workflow process, we integrate all specifications into one Petri Net. After that, the integrated Petri Net can be checked for consistency. *We consider the process and atomicity specifications to be consistent if a scheduler can execute the process definition according to the atomicity constraints.* We introduce *relaxed soundness* as a criterion for this property.

6.1. Informal analysis

As a first step the independently made specifications from Figure 1 and Figure 3 are mapped onto each other which results in Figure 8. Here the workflow process specification and the atomicity spheres are presented in the notation introduced earlier.

From this notation we can see that the strict atomicity sphere around *pick, wrap* and *deliver* is consistent with the process specification. The three tasks are executed in a row and the process specification does not offer the possibility to skip either of the tasks. However, for the alternative atomicity over the { *pick, wrap, deliver*}, { *take_orderC1* } and { *notify_cancel* } spheres this is not obvious. Here, analysis techniques should prove consistency.

The integration of the exception atomicity sphere requires a little more thought as with other spheres, because the exception sphere adds a new exception task to the model, which was not present in the process model before. Therefore, we must include additional process specification to specify what should happen in case the exception is raised. In our example the workflow designer decides that in case of an exception the process should behave as if the customer was checked and found OK (modeled by the arc from *X* to *S7*).

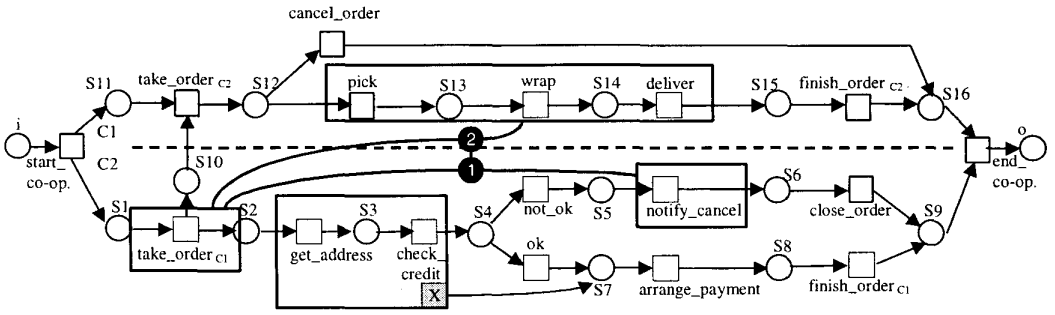

Figure 8. Example with atomicity annotation

6.2. Integrated Petri Net

To prove consistency of the workflow process and the atomicity specifications as presented in Figure 8, we now transform the different models into one single Petri Net. The integration is performed by joining the WF-Net with the atomicity sphere Petri Net patterns. The Nets are merged at the points where they have common transitions. In addition, we extend the Net with a source place (*I*) and a sink place (*O*) and two transitions *initiate* and *clean_up*. Transition *initiate* is introduced to initialize the workflow Net and all atomicity spheres. This is done by connecting it to the new source place *I* and with all source places from the atomicity spheres (*local i*'s) and the source place from the process definition (*i*). The transition *clean_up* is introduced to collect all tokens again from the workflow Net and the atomicity spheres. It connects all sink places (*o* and *local o*'s) with the new sink place *O*. Note, that the integration of the Petri Nets can easily be implemented with an automated tool. This tool would have the WF-Net and the Petri Net patterns as input and would produce the combined Petri Net.

Figure 9 shows the integrated WF-Net after combining the Nets corresponding to the spheres in Figure 8. The original workflow Net is marked in gray. Note that the strict atomicity spheres around task *notify_cancel* and *take_orderC1* are omitted, because we consider a single task to behave strict atomically by itself and therefore requires no specific Petri Net pattern.

6.3. Relaxed soundness as correctness criterion

Van der Aalst introduced *soundness* as a correctness criterion for Workflow Nets [1]. It covers a minimal set of requirements a process definition should satisfy. Soundness ensures that, the process can always terminate with a single token in place *o* and all the other places empty. In addition it requires, that there is no dead task, i.e. all tasks can be executed.

Next we define soundness according to [1]. Note that the notation $M_1 \rightarrow^{t} M_2$ denotes that the firing of transition t brings state M_1 to state M_2. $M_1 \rightarrow^{*} M_2$ denotes that there exists a firing sequence of tasks that brings state M_1 to state M_2.

Definition (Sound). A process specified by a WF-Net $PN = (P,T,F)$ is *sound* if and only if:

(i) For every state M reachable from state i, there exists a firing sequence leading from state M to state o. Formally:

$$\forall M\,(i \rightarrow^{*} M) \Rightarrow (M \rightarrow^{*} o)$$

(ii) State o is the only state reachable from state i with at least one token in place o. Formally:

$$\forall M\,(i \rightarrow^{*} M \wedge M \geq o) \Rightarrow (M = o)$$

(iii) There are no dead transitions in PN with initial marking i. Formally:

$$(\forall t \in T)(\exists M, M')\ i \rightarrow^{*} M \rightarrow^{t} M'$$

A WF-Net that is extended with a transition that connects place *o* with place *i* is sound if and only if the extended Net is live and bounded [1]. Therefore the soundness criterion requires that a WF-Net can never deadlock. To avoid deadlocks, decisions in the Net must be taken in advance, before a deadlock can occur. This would introduce explicit synchronization in the Net. We argued in section 3 that we want to leave this synchronization out of our specification and transfer the responsibility of *avoiding* or *resolving* deadlocks to a scheduler. Therefore we can relax the soundness criterion to a new criterion called *relaxed soundness*.

The intuition of relaxed soundness is that for each transition *there exists a* firing sequence that brings the initial state *i* to state *o*. No tokens should be left in the Petri Net. We call this a *sound firing sequence*. Note that the definition of soundness requires *all* firing sequences to

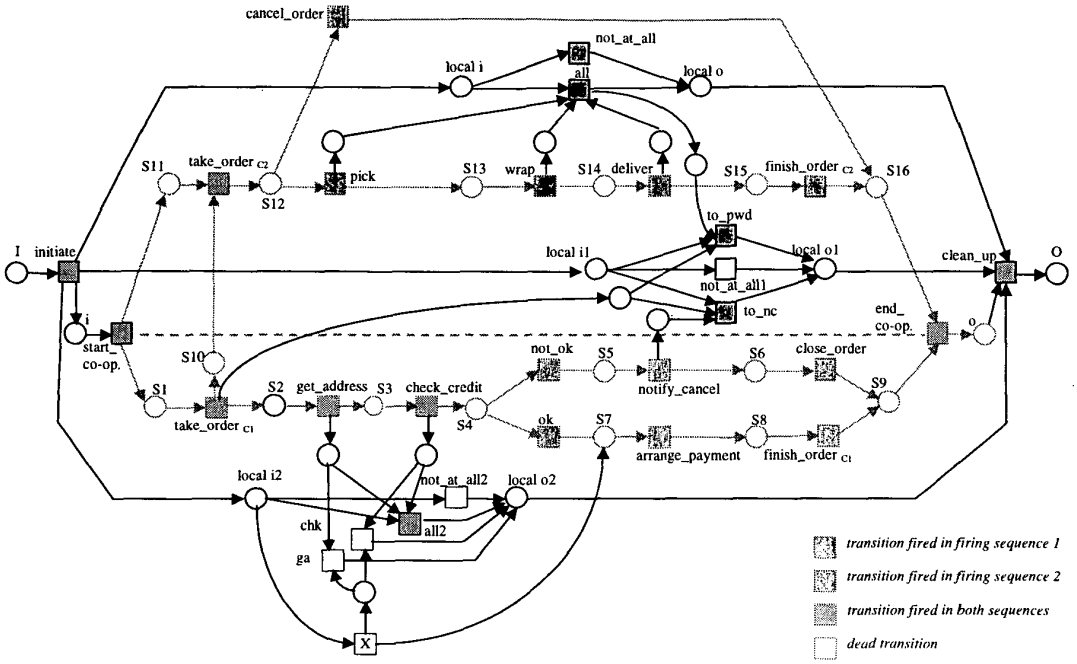

Figure 9. Integrated WF-Net with marked firing sequences 1 and 2

be a sound firing sequence. Therefore soundness implies relaxed soundness.

Definition (Relaxed sound). A process specified by a WF-Net $PN = (P,T,F)$ is *relaxed sound* if and only if every transition t is in a firing sequence that starts in state i and ends in state o. Formally, with M, M' markings of PN:

$$(\forall\, t \in T)\,(\exists\, M, M')\ (i \rightarrow^* M \rightarrow^t M' \rightarrow^* o\,)$$

6.4. Application of relaxed soundness

In this paragraph, we apply the relaxed soundness criterion to check consistency of workflow processes and atomicity specifications. With this build-time analysis of the workflow and atomicity specification, we can determine whether the workflow can execute according to the atomicity requirements. We apply the criterion to the example of Figure 9.

To check for relaxed soundness we have to check whether every transition is in a firing sequence that starts in state I and ends in state O. As there are many transitions, this should be automated by a tool, e.g. by Woflan [11] or LoLa [8]. As an illustration, we analyze the Petri Net by hand in the next section. The approach we take is to find sound firing sequences in Figure 9 that start in state I and end in state O. For such firing sequences we mark the transitions. For each transition that is not marked, we then try to find another firing sequence from state I to state O. If there appears a transition that cannot be marked, the Petri Net is not *relaxed sound.*

As an example, we mark a sound firing sequence that includes task *ok* (ordered item is delivered) and a sound firing sequence that includes task *not_ok* (ordered item is not delivered) in Figure 9.

We see that four transitions *not_at_all1*, *not_at_all2*, *chk*, *ga* and *X* are not included in these firing sequences. So for each of these transitions we have to find another sound firing sequence in order to prove relaxed soundness. However, in the Net there is no firing sequence possible that includes any of these transitions. So, the relaxed soundness criterion is violated for these transitions and thus the Net in Figure 9 is not relaxed sound. This indicates inconsistency of the workflow process specification and the atomicity specification. To resolve the inconsistencies we can adopt two strategies: *change the workflow specification* or *change the atomicity spheres*. Note that this is a design problem and therefore the correction is done manually. In this example we only change the process specification.

For the inconsistency with the *not_at_all1* and *not_at_all2*, we add a transition *no_exec* to the workflow specification that connects the I and O place directly. This way we allow the Workflow Net to execute without doing any work, making the dead transitions of the atomicity spheres live again. To resolve the exception sphere inconsistency, we add skip transitions to the process

specification that allows non-execution of the *get_address* and *check_credit* transition. Note that even if one of the tasks is skipped, always a token will be present in *S4*. Therefore we consume this token if the exception task *X* is executed. Now the Petri Net is *relaxed sound.*

Note that relaxed soundness implies that *every* transition in the Petri Net should be live, including all transitions in the atomicity spheres. This forces the designer to include in the workflow specification all possible executions of the atomicity spheres. For example, in case of the exception sphere, the workflow specification must support all four exception cases. This is ensured by adding the *skip1* and *skip2* tasks. Also, the task *no_exec* is added to ensure the possible firing of transition *not_at_all1* and *not_at_all2*. In case it is not critical that all alternatives of the atomicity spheres are supported by the workflow specification, the relaxed soundness criterion could only be applied to a subset of the tasks.

7. Conclusions and future work

This paper introduces a new approach for specifying atomicity requirements for workflow applications. We propose to separate the specification of the workflow process and the atomicity requirements. This allows the workflow and transaction designers to work independently. Atomicity defines execution dependencies between a group of tasks, independent of ordering. In traditional workflow specifications, these execution dependencies are specified implicitly within the ordering. If recovery is absent, this results in pessimistic execution schedules. Soundness is a correctness criterion that allows pessimistic process specifications only. However, we argue that if the workflow is supported by advanced transaction management that supports recovery, more optimistic schedules should be allowed, because transaction management could recover from potential deadlocks. Therefore we relax the soundness criterion to *relaxed* soundness. Relaxed soundness allows the designer to specify a wider class of workflows and thus a more flexible way of execution.

We have shown how relaxed soundness can be applied to check the consistency of the workflow process and atomicity specification. In addition, we have shown that our example is relaxed sound. Because the example is *relaxed sound*, it can be executed, even though the specification is not *sound.*

In future work we will consider other transaction properties than atomicity, e.g. isolation and recovery. In addition, we will elaborate on transformations between relaxed sound Nets and sound Nets.

Acknowledgements

We thank Wil van der Aalst for inspiring discussions and helpful hints for improving this paper.

References

[1] Aalst, W.M.P. van der; The Application of Petri Nets to Workflow Management, The Journal of Circuits, Systems and Computers 8(1), 1998, pp. 21-66

[2] Alonso, G.; Agrawal, D.; El Abbadi, A.; Kamath, M.; Gunthor, R.; Mohan, C. Advanced transaction models in workflow contexts, ICDE96, New Orleans, 1996

[3] Alonso, G. Processes + Transactions = Distributed Applications, Proceedings of the 7th international Workshop on High Performance Transaction Systems, Asilomar, 1997

[4] Adam, N.R.; Atluri, V.; Huang, W., Modeling and Analysis of Workflows Using Petri Nets, Journal of Intelligent Information Systems 10 (2), 1998, 131-158

[5] Attie, P.; Singh, M.; Sheth, A.; Rusinkiewicz, M.; Specifying and Enforcing Intertask Dependencies, Proceedings of the International Conference on Very Large Databases, Dublin, 1993, pp. 134-145

[6] Grefen, G.; Pernici, B.; Sanchez, G. (eds.); Database support for Workflow Management - The WIDE Project, Kluwer Academic Publishers, Boston, 1999

[7] Leymann, F.; Roller, D. Production Workflow: Concepts and Techniques, Prentice Hall, 2000

[8] Lola, a Low Level Petri Net Analyzer, Humboldt University Berlin, http://www.informatik.hu-berlin.de/~kschmidt/lola.html

[9] Reuter, A.; Schneider, K.; Schwenkreis, F. ConTracts Revisited. In: Jojodia, S.; Kerschberg, L. (eds.); Advanced Transaction Models and Architectures, Kluwer, 1997, pp. 127-151

[10] Sheth, A.; Rusinkiewicz, M.; On transactional Workflows, In: Hsu, M. (ed.), Special Issue on Workflow and Extended Transaction Systems 16, IEEE CS, Washington, 1993

[11] Verbeek, H.M .W.; Aalst, W.M.P. van der; Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool, In: Nielsen, M.; Simpson, D. (eds.); Application and Theory of Petri Nets 2000, LNCS 1825, Springer-Verlag, Berlin, 2000, pp. 475-484

[12] Vonk, J.; Derks, W.L.A.; Grefen, P.; Koetsier, M. Cross-Organisational Transaction Support for Virtual Enterprises. Proceedings of the fifth IFCIS International Conference on Cooperative Information Systems, Eilat, 2000