Customized atomicity specification for transactional workflows

Wijnand Derks[1], Juliane Dehnert[2], Paul Grefen[3], Willem Jonker[1]

[1]KPN Research, {w.l.a.derks, willem.jonker}@kpn.com

[2]Technische Universität Berlin, dehnert@cs.tu-berlin.de

[3]Center for Telematics and Information Technology (CTIT), University of Twente, grefen@cs.utwente.nl

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.

Keywords: transactional workflow, Petri Net, soundness, atomicity

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, it is very hard to develop a single transaction model that fits all transaction requirements in all workflow applications. Typically, transaction models suit specific parts of a workflow application, whereas they do not suit other parts. Otherwise, a transaction model will suit one workflow application, but not another. We claim therefore that transaction requirements for workflows are often application dependent. This implies that for each workflow application, the transactional properties should be included in the specification. Hence, we require a model to express transactional requirements for workflow applications explicitly. This means that we specify transactional properties independently from existing solutions (i.e. transaction models).

We consider the specification of transactional workflow applications to consist of two design tasks: workflow specification and transaction specification. Typically, a workflow designer specifies the workflow specification, whereas a transaction specialist defines transactional requirements. Also, transaction requirements may change while the workflow stays the same and vice versa. Therefore we require the transactional properties to be defined independently of the workflow process.

Eventually, the workflow process and the transaction requirement specification should be checked for consistency. We consider the specification consistent, if the workflow process can be executed according to the transactional requirements. To check consistency of specifications, we rely on a workflow specification model and a transaction specification model that can be integrated into a single model.

The approach we take is to specify both models with Petri Nets. We take Petri Nets, because it is a well-defined formalism for which many techniques and tools exist. For the specification of workflows we adopt existing work on Workflow Nets from [Aalst98b], which are based on Petri Nets. For the specification of transaction requirements we introduce transaction property definitions. From these definitions we proposeimplementations by Petri Net patterns. We present a means to integrate both models. This integrated model can then be checked for consistency..

In this paper, we restrict ourselves to the transactional property *atomicity*. Atomicity constraints describe the existence relation between tasks. A typical example of an existence relation between tasks in a group is that either all tasks in the group should execute, or no task of the group should execute. In most workflow process definitions this property is implicitly specified in the ordering of tasks, because a task is assumed to execute if it is enabled. Hence, a sequence of tasks in a workflow specification typically behaves as an atomic unit. In case the tasks have no direct ordering relationship, e.g. with parallel processes, the atomic behaviour is not obvious. Note that this is especially the case in cross-organisational workflows where two processes agree on atomic execution of tasks (e.g. payment and delivery). Separating the specification of atomicity and ordering preserves functional semantics and does not include the atomicity specification implicitly in the ordering in terms of synchronization. For the specification of the atomicity requirements, we introduce an intuitive and simple notation.

To verify whether the workflow process can be executed meeting the atomicity requirements, we merge the two models into an integrated model. We provide rules for the translation of the atomicity requirements into Petri Nets and describe the combination of the different Nets. 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 [Aalst98b]. 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. Here the participating companies independently specify their individual workflows with ordering relations on their tasks, while the agreements of the co-operation are expressed as existence relations over groups of tasks. The need for an independent specification of ordering and atomicity semantics becomes therefore especially clear.

The rest of this paper is organized as follows. In Section 2, we describe how our approach relates to existing work. In Section 3, we introduce Workflow Nets as our workflow specification formalism. We introduce an example Workflow Net that contains atomicity requirements . In Section 4, we introduce atomicity spheres. After that, we show in Section 5, how to translate the atomicity spheres into Petri Nets. In Section 6, we combine the workflow process definition and the atomicity spheres into a single Petri Net. To check consistency of both models we introduce *relaxed soundness* as a correctness criterion. We show how this relaxed soundness criterion applies to our example. Finally, we discuss our conclusions and future work in Section 7.

2 Related work

The traditional transactions that have strict ACID properties [GR93] are not suitable for long running transactions. A reason for this is that the strict atomicity requirement would imply too much work to be undone in case of failure. In addition, strict isolation imposes too severe restrictions on concurrency and even prohibits co-operation. Therefore, advanced transaction models have been developed that relax the atomicity and isolation requirements. In the late eighties and early nineties, many advanced transaction models were developed, of which important ones are included in the work [Elmag92] and [JK97]. In the nineties workflow applications were identified as an important application domain for transaction management. In [SR93] 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.

Important approaches in the literature include:

1. modeling of transactional properties, e.g. [Chrys91];
2. specification, analysis and support of transaction state dependencies, e.g. [Reu89], [ASSR93], [GH94], [GHM96], [TV98], [AAH98];
3. integration of advanced transaction models, e.g. [CR94a], [GPS99];
4. application of existing advanced transaction models to a workflow, e.g. [MAAE+95], [KMO98], [GPS99], [VDGK00],
5. specifying spheres of control, e.g. [Davie78], [Leyma95], [LR00], [Alons97]

We discuss these approaches below.

2.1 *Modeling of transactional properties*

Work on modeling transactional properties is rather limited and the best known approach is ACTA, which was developed by Chrysanthis [Chrys91]. ACTA facilitates the formal description of properties of advanced transaction models and is described with first-order logic. The framework is based on five building blocks: history, dependencies between transactions, the view of a transaction, the conflict set of a transaction, and delegation. Each transaction submits significant events, i.e. initiation events (e.g. begin, split) and termination events (e.g. commit, abort, join). Conditions on the event history define what schedules are allowed. Dependencies between transactions can be expressed by logical implications on the significant events of transactions. For example, the abort event of one transaction may imply the begin of another transaction. These dependencies arise either from structure (e.g. parent-child relationship in nested transactions) or from behavior (e.g. concurrent access of an object). The view of a transaction (i.e. visibility) defines to what extent another concurrent transaction may see the results *before* it has committed. The notion of conflict set defines what operations of transactions in the history should be considered when scheduling of a new operation. Delegation transfers the responsibility of terminating a transaction to another transaction instead of the invoker. Thus ACTA defines a formal framework for the execution of a specific transaction model.

Our approach is different from ACTA because we do not consider the properties of a certain transaction model, but rather consider the transactional requirements for a specific workflow. We therefore start at the requirements level and will eventually map our requirements onto a transaction model. The ACTA approach would be to develop a generic transaction model and then apply it to a workflow specification. This is rather the other way around. Also, ACTA defines the transaction model, but does not include workflow specification within the model. Hence it does not provide a means for integrating both specifications.

2.2 Transaction state dependencies

To close the gap between transaction models and applications, Reuter [Reute89], [Reute97] introduces a transactional workflow specification and execution framework called ConTracts. It supports execution of (a group of) tasks that preserve strict ACID properties and is called an atomic unit in case of a group of tasks. The control flow between tasks is defined based on the abort or commit of a transaction. The management of control flow and recovery is managed by the ConTract system.

Compared to our approach, the ConTract framework assigns strict atomicity and isolation for atomic units, whereas in our work atomicity is considered a separate property. In addition, abort dependencies define recovery explicitly in the workflow model, which is contradictory to our requirement for separating workflow and transaction specification. Hence, the ConTract provides a control mechanism above ACID transactions, but does not provide a means to define customized transaction properties over multiple tasks.

Influenced by the work of ACTA, [ASSR93] proposes a more refined transaction state model of an activity than the ConTracts system. A task can be non-executing, executing, done, aborted and committed. Inter-task dependencies both describe the workflow as well as the transactional

properties. Model checking is performed by mapping the inter-task dependencies to Computation Tree Logic (CTL) formulae. The focus of this work is on enforceability of the inter-task dependencies. For this, the significant events cm, ab, pr, st are categorized whether they are forcible, rejectable or delayable. For each dependency, a state transition diagram is composed, which is known by the scheduler. The scheduler can then calculate what dependencies can be guaranteed at run-time by calculating *viable path-sets*. In [TV98], it is argued that the approach of state transition diagrams results in a specification explosion and therefore should be improved in order to be practically applicable. Therefore, the authors develop an algorithm that prunes the state automata. Their approach scales linearly.
Similar to the ConTracts approach, [ASSR93] specifies transactional behavior within the workflow specification, whereas we separate both specifications.

In [GH94], the model of transaction state dependencies is extended with correctness dependencies. Similar to [ASSR93], transaction state dependencies cover both workflow and atomicity specifications. In relation to our approach, the transaction state dependencies in this work again do not distinguish between workflow specification and transaction properties.
In [GHM96] an infrastructure is proposed that supports the specifications defined in [GH94]. Also, a framework is introduced that can determine whether an extended transaction model (ETM) can be supported. Note that the approach of specification is similar to our approach, but transaction and workflow specification are again combined, while we explicitly separate them.

[AAH98] builds further on the approaches of [ASSR93] and [GH94] and offers a formal framework for specification of transactional workflow applications based on Petri Nets. The activities of the workflow consist of different states similar to [ASSR93] (e.g. initialized, executing, done, aborted, committed). Transactional properties between activities are expressed by inter-task control flow dependencies: precedence and causal. The precedence type defines the order in which states of two tasks may occur and the causal type defines what combinations of states may or must (not) occur. The causal types define existence relations and are expressed by logical implications. Workflow specification is defined by precedence order dependencies as well as multiple dependency control flow dependencies (i.e. or, and, fork, split).
Hence, similar to [ASSR93] transactional properties between activities are expressed *implicitly* by operational inter-task dependencies, whereas we specify them explicitly by separating them from the workflow specification. Similar to our approach, they show how the inter-task dependencies

can be specified in terms of Petri Nets and analyze the Petri Net on inconsistent dependencies and safety.

2.3 Integration of advanced transaction models

In [CR94a], it is shown how the ACTA framework can be applied to integrate different transaction models into a single new transaction model. Examples in this paper include variations on the joint-transaction model [Pu88]: reporting transactions and co-transactions. Also variations on the nested-transaction model and the split-transaction model [Pu88] are discussed. Note that the specifications of the transaction models are axiomatic and define rules for execution of the transaction models.

Our approach of first specifying the required transactional properties and then find a suitable transaction model is supported by this work. [CR94a] shows that different transaction models can be combined to suit the specified transactional properties. Note that the work does not mention the relation with workflows, whereas we include it.

In the WIDE project [GPS99], a workflow is supported at two transaction levels: global and local. The global transactions are implemented by extended SAGA's and the local transactions are implemented by nested transactions. At the global level, the SAGA-based model offers relaxed atomicity through compensation and relaxed isolation by limiting the isolation to the SAGA steps. The SAGA steps themselves are mapped onto a single nested transaction. This way, the ACID properties of the SAGA steps have a direct match with the nested transactions. At this finer granularity the workflow activities are defined and therefore the grouped workflow activities follow the strict ACID properties. Note that this approach provides flexibility in assigning transaction properties to workflow activities, because workflow activities can be grouped as desired. However, the flexibility is limited to the extended SAGA or nested transaction model.

2.4 Application of advanced transaction models

The Exotica project [MAAE+95] aimed at exploring several research areas in order to improve on the IBM Flowmark product. The main research issues included fault-tolerance, scalability, mobile and disconnected computing, and advanced transaction management. In [AAEK+96], the advanced transaction management is discussed. They show how the SAGA model and the Flexible Transaction models can be modeled in a Flowmark process specification. So, the transaction model and workflow are integrated into a single workflow specification. This means, that retries and compensation flows and decisions are all incorporated in the workflow specification. To overcome this design complexity for the SAGA model, the product FlowBack [KMO98] was developed. This product generates the compensation graphs from the workflow specification according to the SAGA model at specification time. Note that in WIDE [GPS99] a similar approach was taken, but here the compensation graphs were calculated dynamically and thus completely left out of the workflow specification. In the ESPRIT project CrossFlow the WIDE approach was extended to cross-enterprise workflows by distinguishing transaction semantics on three cross-organisastional levels [VDGK00]. Specifically, compensation was extended to compensation for groups of tasks and compensation performed by the other organisation than the original performer.

2.5 Spheres of control

The concept of *spheres of control* was introduced by Davies [Davie78]. Davies distinguishes multiple data processing spheres of control to ensure correct data processing. The spheres include: process control, recovery control, auditing control and relational integrity control. The process control spheres are related to our work. Process control spheres define levels of processing, where each level's implementation is transparent to the level above. Process atomicity is a property of each level and refers to the amount of processing that one wishes to consider as having identity. Process commitment control determines the containment of the effects of a process, enabling a rollback in case the processing should be restarted. Controlled dependency records the tasks that rely on each other's outcomes. They facilitate cascaded aborts. Further, Davies also distinguishes in-process and post-process recovery spheres. In-process recovery spheres allow to return a process to a previously acceptable point and post-process recovery allows to search back to find the source of error in multiple finished processes. Although Davies' ideas are very much related, he only presents his ideas in an informal manner, whereas we define transaction properties with Petri Nets.

[Davie78] shows that the idea of spheres of control are quite old, but only recently these ideas are applied to workflows. Particularly, Leymann [Leyma95] follows similar ideas of spheres of control arguing that existing transaction models like closed nested transactions, open nested transactions and SAGA's are not sufficient in workflow applications because of their rigorous recovery characteristics, i.e. all work will be undone in case of some failure in a workflow. In [Leyma95] he defines *spheres of joint compensation* and argues that they should be included in the workflow specification to allow for customized and hence more efficient recovery. In [LR00] this work is extended with the concept of *atomic spheres*. Atomic spheres define groups of tasks that jointly commit or abort. The author proposes operations to be included in the IBM MQ Workflow product.

In this paper we do not consider compensation, but the spheres of compensation can be included to our workflow specification to recover from process states that would violate the atomicity requirements. An *atomic* sphere specifies that the results of the contained tasks be made persistent (all commit), or all results be discarded (all abort). Hence, [LR00] approaches atomicity from a persistence perspective. This is different from our approach, because we define atomicity as a property of the workflow. We use atomicity to prescribe what groups of tasks should either all execute or none. Persistence can be specified in addition to this atomicity specification.

In [Alons97] Alonso argues that transactions should be applied to processes. In particular, he expresses the need for light-weight transactions and the need to express separate transactional semantics to be applied to groups of activities. He distinguishes spheres of atomicity, isolation and persistence. The sphere of atomicity includes the atomicity requirements of activities but also the recovery behaviour. Here an activity could be declared basic (non-atomic), semi-atomic, atomic, restartable and compensatable. However, Alonso does not provide formal specification of atomicity. Also, atomicity and recovery are coupled, whereas we consider atomicity separately from recovery. In [HA98] an overview of the OPERA system is presented in which the atomicity, isolation and persistence spheres are mentioned. However, the implementation is based on the flex transaction model and does not provide means to specify atomicity, isolation and persistence independently from the flex model.

3 Workflow Process

In this section we introduce the specification for the workflow processes. For this purpose we adopt the Workflow Nets as introduced by Van der Aalst [Aalst98b]. We use an example from the logistics domain to show the application of the Workflow Net. In the second subsection we motivate the need for atomicity spheres.

3.1 Workflow Nets

A workflow process consists of tasks that are related by control flow, data or external dependencies. For complexity reasons we consider in this paper only the control flow dependencies. In this perspective, workflow process definitions are defined to specify which tasks need to be executed and in what order (including no order).

For the specification of a workflow process definition we use Petri Nets, because Petri Nets have a clear and precise definition, e.g. [Aalst98b], [Mur89], [Rei85]. In addition, Petri Nets allow us to make use of many existing analysis techniques and tools. Van der Aalst applies Petri Net theory to workflow specification and introduces Workflow Nets (WF-Net). A WF-Net is a Petri Net, which has a unique source place (i) and a unique sink place (o). A workflow 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 addition, a WF-Net requires all nodes (i.e. transitions and places) be on a path from i to o. This ensures that every task (transition) and every condition (place) contributes to the processing of cases. In this paper we take the definitions of a Petri Net and WF-Net from [Aalst98b]:

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

A WF-Net is a special Petri Net, defined as follows:

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. The process of company C2 is defined above the dotted line and the process of company C1 is defined below the dotted line. 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. We could omit them by connecting the arcs from S4 directly to *notify_cancel* or *arrange_payment*.

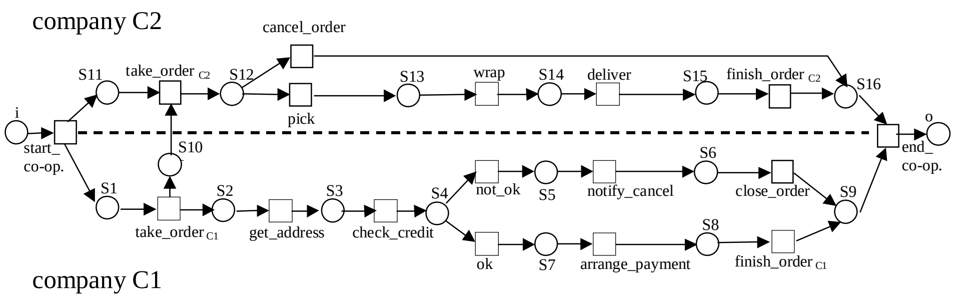


Figure 1 : WF-Net order processing

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 (see also [LG00]).

3.2 Functional synchronization

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 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 [VDGK00].

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

In this section, we introduce the definition of *atomicity spheres*. In addition, we provide an intuitive and simple notation.

4.1 Atomicity types

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. Two exclusive alternatives should be defined here: { *take_orderC1*, *cancel_order* } and { *take_orderC1*, *finish_orderC1* }. 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. Considering the example in Figure 1 it may be possible that one of the tasks *get_address* and *check_credit* did not execute. In case of partial execution, an exception task is executed which initiates the handling of the atomicity violation. In our example, the exception task could handle the violation and could proceed the process as if the customer was checked in a normal way. 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.

An atomicity sphere is drawn as a solid box containing the corresponding tasks. Alternatives are denoted by dark bullets that connect atomicity spheres. The bullets themselves are also connected. Exceptions are visualized by a gray exception task. In case of exception-atomicity, the gray exception task is drawn in a corner of the sphere. For alternative exception atomicity, the gray exception task connects the dark bullets. The notations of the different atomicity constraints are depicted in Figure 2 for some examples with six tasks.

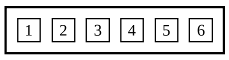
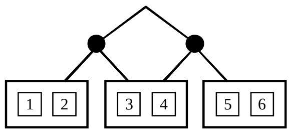
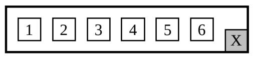
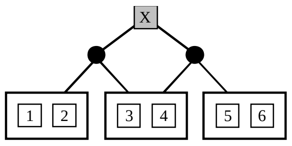
	strict (s)	alternative (a)
no exception (n)		
exception (e)		

Figure 2 : The notations for the different atomicity types

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. A valid execution would be that all tasks 1 through 6 execute, including the exception task.

Figure 3 shows the truth tables for all four examples. For simplicity, we have omitted the tasks 2, 4 and 6 because the strict atomicity spheres require them to execute similar to tasks 1, 3 and 5 respectively. In the left columns the zero means no execution and the one means execution. Note that the X task is only defined for the exception spheres. In the sphere columns, a one represents a valid combination and a zero represents an invalid combination.

	1	3	5	X	strict (s)	alternative (a)
no exception (n)	0	0	0	-	1	1
	0	0	1	-	0	0
	0	1	0	-	0	0
	0	1	1	-	0	1
	1	0	0	-	0	0
	1	0	1	-	0	0
	1	1	0	-	0	1
	1	1	1	-	1	0
exception (e)	0	0	0	0	1	1
	0	0	0	1	0	0
	0	0	1	0	0	0
	0	0	1	1	1	1
	0	1	0	0	0	0
	0	1	0	1	1	1
	0	1	1	0	0	1
	0	1	1	1	1	0
	1	0	0	0	0	0
	1	0	0	1	1	1
	1	0	1	0	0	0
	1	0	1	1	1	1
	1	1	0	0	0	1
	1	1	0	1	1	0
	1	1	1	0	1	0
	1	1	1	1	0	1

Figure 3 : Truth tables for the different atomicity types

4.2 Application in the example

To demonstrate the use of atomicity spheres, we have identified atomicity requirements in the example of Figure 1. The spheres are depicted in Figure 4.

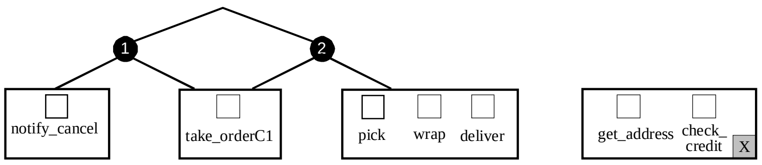


Figure 4 : Atomicity specifications in the example

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. This exception task then initiates the handling of the atomicity violation. In Section 6.1 we show how task *X* is integrated in the process specification.

5 Specifying atomicity with Petri Nets

In this section, we map the atomicity spheres to Petri Net patterns. Recall that we want to integrate the atomicity specification with the workflow specification and therefore specify both in the same formalism.

5.1 Strict Atomicity

An atomicity-sphere containing a group of tasks specifies the behavior that the tasks should either execute all or none. The following picture gives the translation of the strict atomicity sphere with two tasks into a Petri Net. This corresponds to the s/n combination in Figure 2 for two tasks.

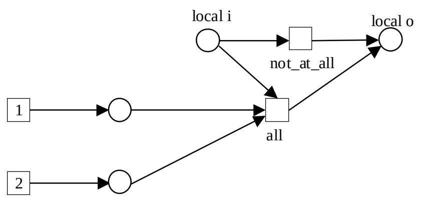


Figure 5 : Petri Net pattern for a strict atomicity sphere

The tasks *1* and *2* in the sphere are present on the left of the graph. They may be activated by the workflow process. The Petri Net has a *local i* and a *local o*. To ensure that the sphere is satisfied exactly once, one token is placed in *local i* only during the initialization of the sphere. Note that this implies that we require the workflow process to be acyclic. The transitions between the *local i* and *o* represent the alternatives of the atomicity sphere that are allowed. In this case this is the *not_at_all* (no task) execution or the *all* (all tasks) execution, which corresponds to the semantics of a strict atomicity sphere.

Remark that in case the *not_at_all* transition and task 1 or 2 fires, the Net ends with those tokens still in the Net. Therefore we incorporate the requirement that the Net should always end with a single token in the end place *o*. We have incorporated this requirement in our correctness criterion *relaxed soundness*, which is introduced in Section 6.3.

5.2 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 6.

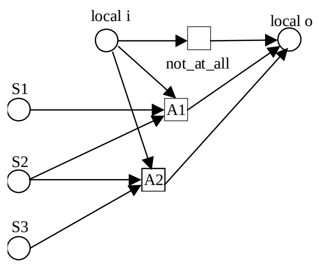


Figure 6 : Petri Net pattern for alternative atomicity

Three alternatives are allowed by the specification: execution of spheres S1 and S2, execution of spheres S2 and S3 or no execution at all. These three cases are represented by the transitions *A1*, *A2* and *not_at_all* between the *local i* and *local o*. The S1, S2 and S3 places directly map onto the *all* transitions of the strict atomicity spheres {1,2}, {3,4} and {5,6}. We have not shown the Petri Net patterns for each of three spheres in the figure for reasons of clarity. In the last paragraph of this section, we give the complete Petri Net for a complex example with six tasks.

5.3 Exception-Atomicity

A strict exception atomicity sphere allows tasks to violate the strict atomicity requirement of all-or-nothing. However, if the strict atomicity requirement is violated, then an exception task is executed. The following picture gives the translation for the example in the e/s-combination (see Figure 2) into a Petri Net pattern for two tasks.

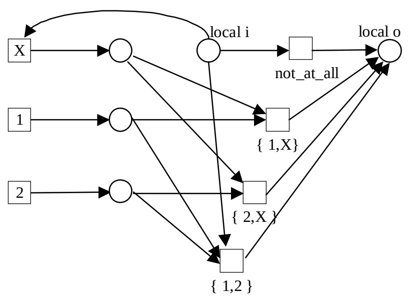


Figure 7 : Petri Net pattern for an exception atomicity sphere

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. For a two-task exception atomicity sphere we have drawn this pattern in Figure 8.

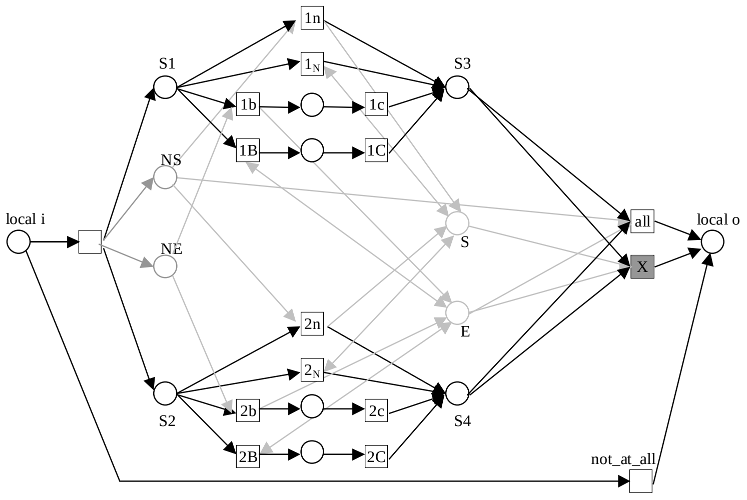


Figure 8 : Petri Net pattern for an exception-atomicity sphere that scales linearly

Every task is now specified with six transitions. Two transitions allow the task to be *skipped* (*Tn* and *TN*) and two transition *pairs* (*Tb*, *Tc* and *TB* and *TC*) specify the *execution* of the task. Furthermore, two pairs of places (*¬Skip, Skip*) and (*¬Exec*, *Exec*) are inserted, represented in the picture as (NS,S) and (NE, E) respectively. They are used as flags, which show whether at least one task was skipped and at least one tasks was executed. The sphere is concluded with transition *all*, if in the end the places *¬Skip* (no task was skipped) and place *Exec* (at least one task was executed) are marked. The sphere is concluded with transition *X* (depicting the exception behavior), if place S*kip* (at least one task was skipped) and place *Exec* (not all task where skipped) are marked.

Note that we changed the modeling of the tasks. A task is now represented by two transitions *Tb* and *Tc*, which denote *begin* and *complete* respectively. We have introduced this, to avoid the *Skip* and *Exec* places to act as a semaphore, which would force sequential execution of the tasks in the Net. So we model the duration of the task explicitly in a place and assign no duration to the *begin* and *complete* transitions. Hence we synchronize only at the *begin* transition. This way of modeling is also according to Van der Aalst [Aalst98].

For example, when the exception sphere is enabled, places NS and NE contain a token, and so have the places S1 and S2. Suppose task 2 begins, then transition 2b fires and moves the token from NE to E. If now task 1 begins, transition 1B consumes the token from E and puts it back after the transition. Eventually, the Petri Net ends with a token in S3, S4, NS and E, so transition *all* will fire. In case transition *1B* did not fire, but transition *NS* did, then the marking would have become S3, S4, S, E and the exception transition *X* would have fired.

The pattern in Figure 8 scales linearly, because for each transition, a fixed number of transitions (i.e. six) and arcs (i.e. 15 black, 8 gray) are incorporated in the net. The linear model results in fewer places and transitions already from six tasks on.

5.4 Exception alternative atomicity

The Petri Net pattern of the exception alternative (e/a) combination is similar to the pattern for the exception atomicity sphere. We have drawn the Petri Net pattern for two spheres S1 and S2, and allow only one of both spheres to execute. In case both spheres execute, an exception is raised. The translation for the example into a Petri Net is given below.

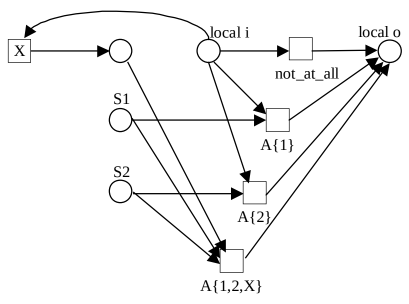


Figure 9 : Petri Net pattern for exception alternative atomicity

The Petri Net incorporates all possible alternative executions of the tasks, i.e. no execution (*not_at_all*), only sphere S1 executes (*A{1}*), only sphere 2 (*A{2}*), and both sphere S1 and sphere S2 execute (*A{1,2,X}*). The execution of both spheres violates the alternative atomicity requirement and therefore results in the execution of the exception task. The exception is modeled by the *X* transition.

5.5 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 10.

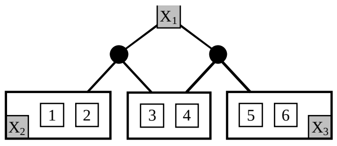


Figure 10 : Example of a complex atomicity constraint

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 with the sphere places Sx of the alternative atomicity Petri Net.

We will show the construction of the Petri Net for the example presented in Figure 10. Note that the construction is straightforward and can easily be automated.

First, the patterns of the atomicity spheres are drawn in Figure 11. Then the pattern for the alternative atomicity is given in Figure 12. The transitions that represent the possible executions of the spheres are connected to the places of the alternative atomicity pattern. The resulting pattern is presented in Figure 13.

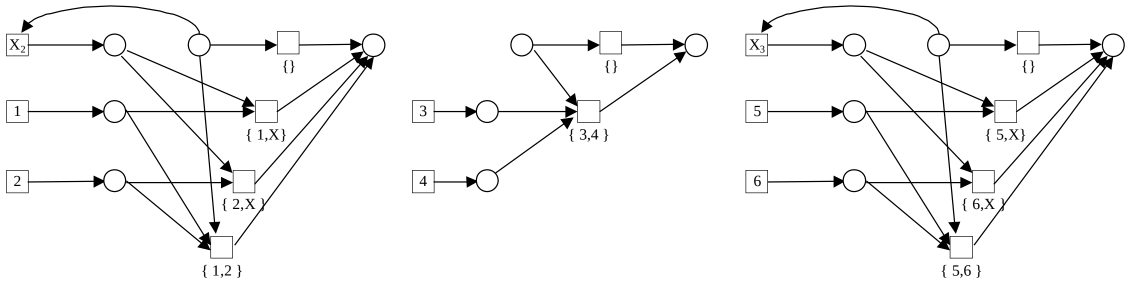


Figure 11 : Petri Net patterns for the atomicity spheres

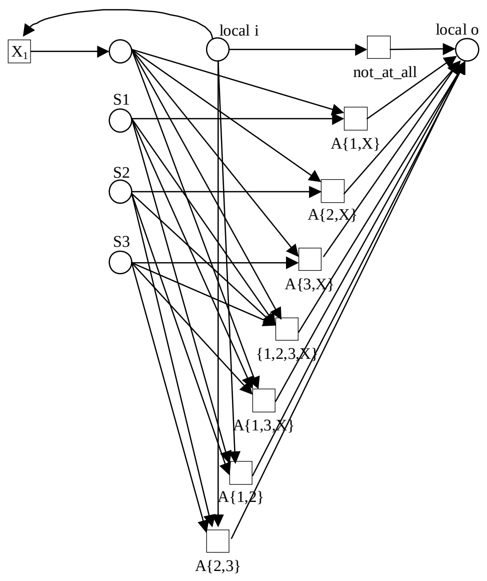


Figure 12 : Petri Net pattern for the alternative atomicity

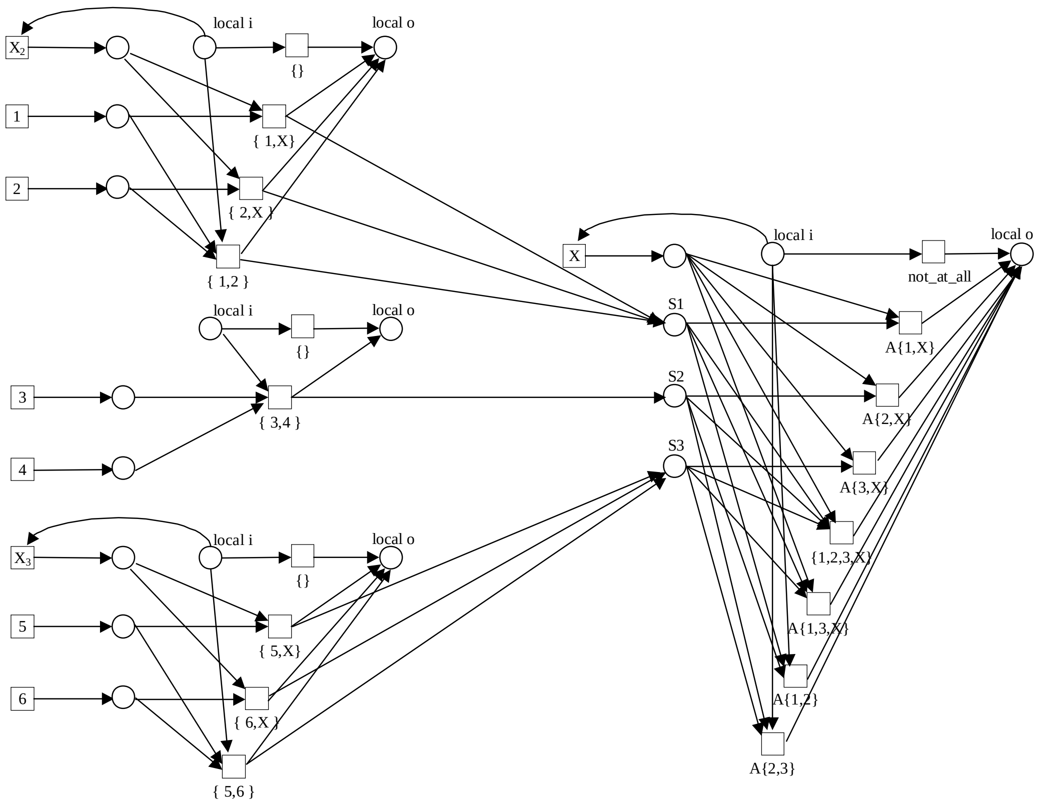


Figure 13 : Integrated Petri Net pattern

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.

The sequel of this section is organized as follows. In the first paragraph, we integrate the notations to give an overview of the complete specification. Subsequently, we integrate the WF-Net from Figure 1 with the Petri Net patterns corresponding to the atomicity spheres from Figure 4. In the subsection after that, we introduce the new correctness criterion *relaxed soundness,* which defines consistency of the specification. In the last paragraph, we show how this criterion applies to the integrated model of the running example.

6.1 Informal analysis

As a first step the independently made specifications from Figure 1 and Figure 4 are mapped onto each other which results in Figure 14. Here the workflow process specification and the atomicity spheres are presented in the notation introduced earlier.

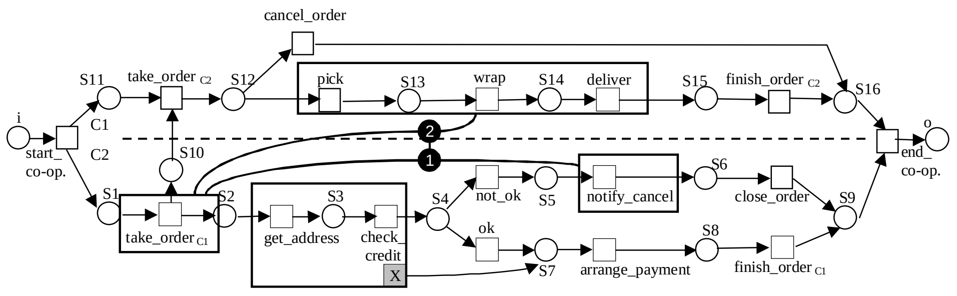


Figure 14 : Example with atomicity annotation

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. The exception is raised in case the customer check was not completed in full, e.g. because it was not possible in a predetermined time interval to establish a database connection. In our example the workflow designer decides that even though the customer check was not finished completely, the process should behave as if the customer was checked and found OK (modeled by the arc from *X* to *S7*).

6.2 Integrated Petri Net

To prove consistency of the workflow process and the atomicity specifications as presented in Figure 14, 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 effect, all arcs in both Nets connected to common transitions are then connected to the combined transition.

Note that the integration simply comes down to merging the transitions of both specification. Hence, 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.

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*.

Figure 15 shows the integrated WF-Net after combining the Nets corresponding to the spheres in Figure 4. 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.

In the next paragraph, we will consider the consistency of the integrated Petri Net.

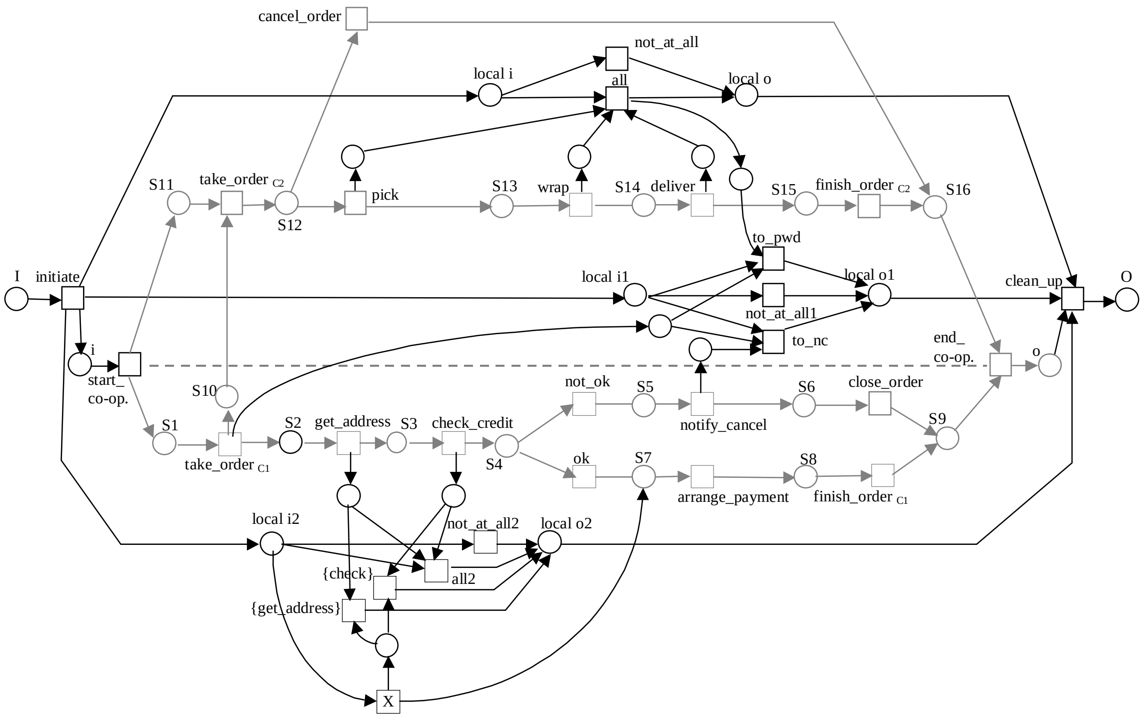


Figure 15 : WF-Net with integrated workflow and atomicity specification

6.3 Relaxed soundness as correctness criterion

Van der Aalst [Aalst98b] introduced *soundness* as a correctness criterion for Workflow Nets. 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 [Aalst98b]. 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:

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

In [Aalst97] it is shown that 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. 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 subsection 3.2 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 want to 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 be a sound firing sequence. We formalize this notion of relaxed soundness by applying the *soundness* requirements (i) and (ii) to a single firing sequence only:

I. There is a marking M_S reachable from state *i* with a firing sequence leading from state M_S to state *o*. Formally:

$$\exists\, M_S\ (i \rightarrow^* M_S\ ) \wedge (M_S \rightarrow^* o)$$

II. State *o* is reachable from state M_S. Formally:

$$\exists\, M\ (\ i \rightarrow^* M_S \rightarrow^* M\ \ ) \wedge (\ M = o\ )$$

Now we can define *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*:

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

Intuitively, relaxed soundness means that a WF net can be executed starting with one token in *i* and ending with one token in *o* and that each transition is involved in some execution of the WF net. Note that soundness implies relaxed soundness.

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 15. We show that the specification is not relaxed sound and that the process and atomicity specifications are indeed not consistent. Then we change the specification and show that this specification is relaxed sound. In addition, we show that the final Net is relaxed sound, but not sound.

To check for relaxed soundness of the integrated Net from Figure 15 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. Note that there are many tools available to check Petri Nets. An example is Woflan [VA00], developed at the University of Eindhoven. Woflan can verify soundness. Another example is LoLA (a **Lo**w **L**evel Petri Net **A**nalyzer), that has been implemented at the Humboldt University of Berlin [LoLa]. It can among others analyze for reachability of a given state, and find dead transitions. Recently, LoLA has been extended to prove for extended computation tree logic-formulas (eCTL) [Roch00]. Within eCTL, it is possible to quantify not only over states, but also over state transitions. The combination of Petri Nets and eCTL allows to check for relaxed soundness: for each transition *t* the reachability of the end-state *o* is verified, while it is required (by eCTL formulae) to include transition *t* in the path to end-state *o*.

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 15 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*.

For the example, we start with the firing sequences as presented in Table 1 and Table 2.

Table 1 : Sound firing sequence 1

fired transition	description
initiate,	activate the workflow specification and the atomicity spheres
start_co-op,	workflow process - starting co-operation
take_order$_{C1}$*, take_order*$_{C2}$*, pick, wrap, deliver, finish_order*$_{C2}$*,*	workflow process at C2 – deliver parcel
get_address, check_credit, ok, arrange_payment, finish_order$_{C1}$*,*	workflow process at C1 – customer is OK and finish order
end_co-op,	workflow process - finish co-operation
all,	strict atomicity sphere - all execute
to_wpd,	alternative atomicity sphere - alternative 2 {take_orderC1, wrap, pick, deliver} executes
all2,	exception atomicity sphere - no exception
clean_up	close activation of workflow and atomicity spheres

Table 2 : Sound firing sequence 2

fired transition	description
initiate,	activate the workflow specification and the atomicity spheres
start_co-op,	workflow process - starting co-operation
take_order$_{C1}$*, get_address, check_credit, not_ok, notify_cancel, close_order,*	workflow process at C1 - customer is not OK, so notify customer about cancel
take_order$_{C2}$*, cancel_order,*	workflow process at C2 - cancel order and do not deliver parcel
end_co-op,	workflow process - finish co-operation
not_at_all,	strict atomicity sphere - no execution because the order is cancelled
to_nc,	alternative atomicity sphere - alternative 1 { *take_orderC1, notify_cancel*} executes
all2,	exception atomicity sphere - no exception
clean_up	close activation of workflow and atomicity spheres

Note that the firing of the transition *ok* represents the case the ordered item was delivered. Similarly, the firing of the transition *not_ok* represents the case where the ordered item is not delivered. Both firing sequences are depicted in Figure 16 with different patterns in the transitions.

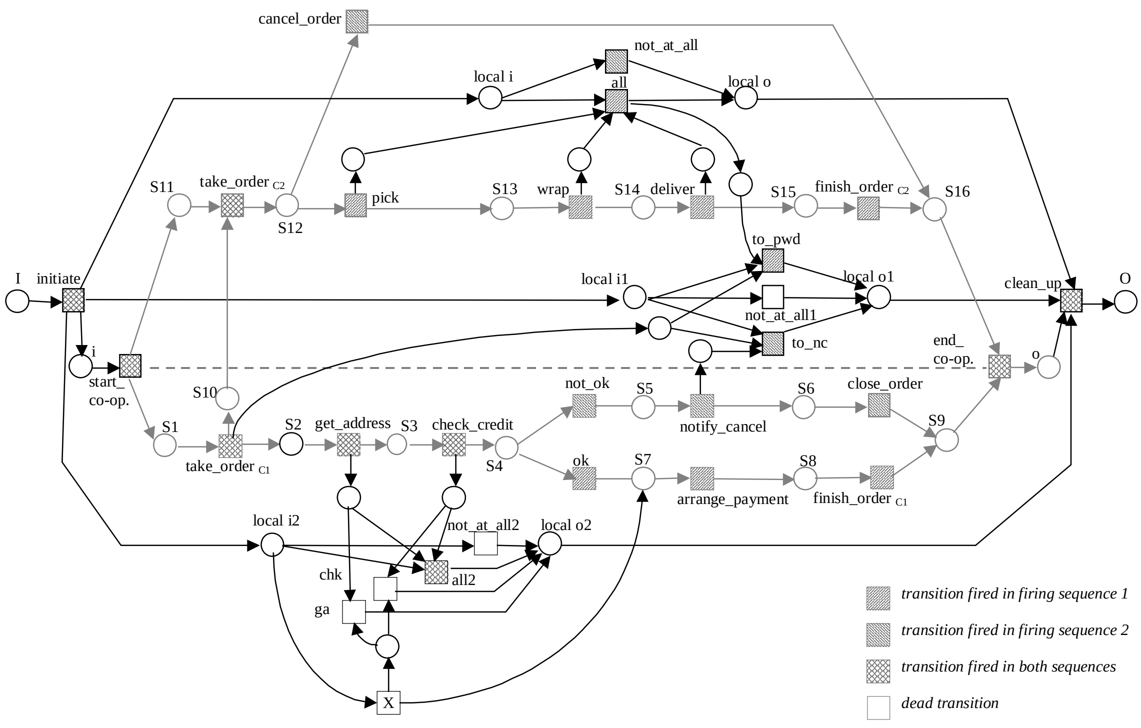


Figure 16 : Integrated WF-Net with marked firing sequences 1 and 2

Now we have marked the transitions fired in the two firing sequences 1 and 2, 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 16 is not relaxed sound. This indicates inconsistency of the workflow process specification and the atomicity specification. We will look at each of the unmarked transitions to identify and resolve the inconsistency.

The transition *not_at_all1* belongs to the alternative atomicity specification, which allows either the execution of tasks {*take_orderC1*, *pick*, *wrap*, *deliver*}, {*take_orderC1*, *notify_cancel*} or none of the tasks. If we look at the process specification of Figure 1 we see, that in all cases the *take_orderC1* task is executed. This contradicts the atomicity case of no execution. The same goes for the *not_at_all2* transition in the exception atomicity sphere for the customer check. Here the process specification implies that the *get_address* and *check_credit* tasks are always executed and thus, the *not_at_all2* transition will never fire in a sound firing sequence. So, the process specification and atomicity specification are inconsistent with respect to the no execution of the tasks in the spheres.

The other transitions *chk*, *ga* and *X* are all part of the exception atomicity sphere and cover the firing sequences in case only one of the tasks *get_address* or *check_credit* are executed. Indeed, in the process specification of Figure 1 we see that the process specification implies that both tasks will always be executed. So, the process specification is inconsistent with the exception cases of the exception atomicity sphere.

To resolve the inconsistencies we can adopt two strategies: *change the workflow specification* or *change the atomicity spheres*. 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.

The resulting Net is shown in Figure 17. In the figure we have also marked each task that is contained in a sound firing sequence. No transitions are unmarked, so 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 cases of execution. 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. Particularly, relaxed soundness would be verified for the whole Petri Net, but only the workflow tasks are verified for liveness. This would allow certain transitions in atomicity spheres to be dead.Finally we will show that the Petri net of Figure 17 is not sound. To prove that the Net in Figure 17 is not sound, we have to provide a marking that violates requirement (i) or (ii) of the definition. An example for this is the marking that results from the firing sequence {*initiate*, *not_at_all*, *start_co-op*, *take_orderC1*, *take_orderC2*, *pick* } which starts from state *I*. The resulting marking has one token in each of the places { *local o*, *S2*, *S_pick*, *S13*, *local i2*, *local i1*, *S_toC1* }. We see that *S_pick* contains a token and this token can only be removed if transition *all* fires. However, transition *all* cannot fire anymore, because the transition *not_at_all* already consumed the token from *local i*. Hence we have ended up in a deadlock. So, even if a token reaches place *O* and hence fulfilling requirement (i), then still there will be the token in *S_pick*, which violates requirement (ii). Therefore the Petri Net from Figure 17 is *not sound*.

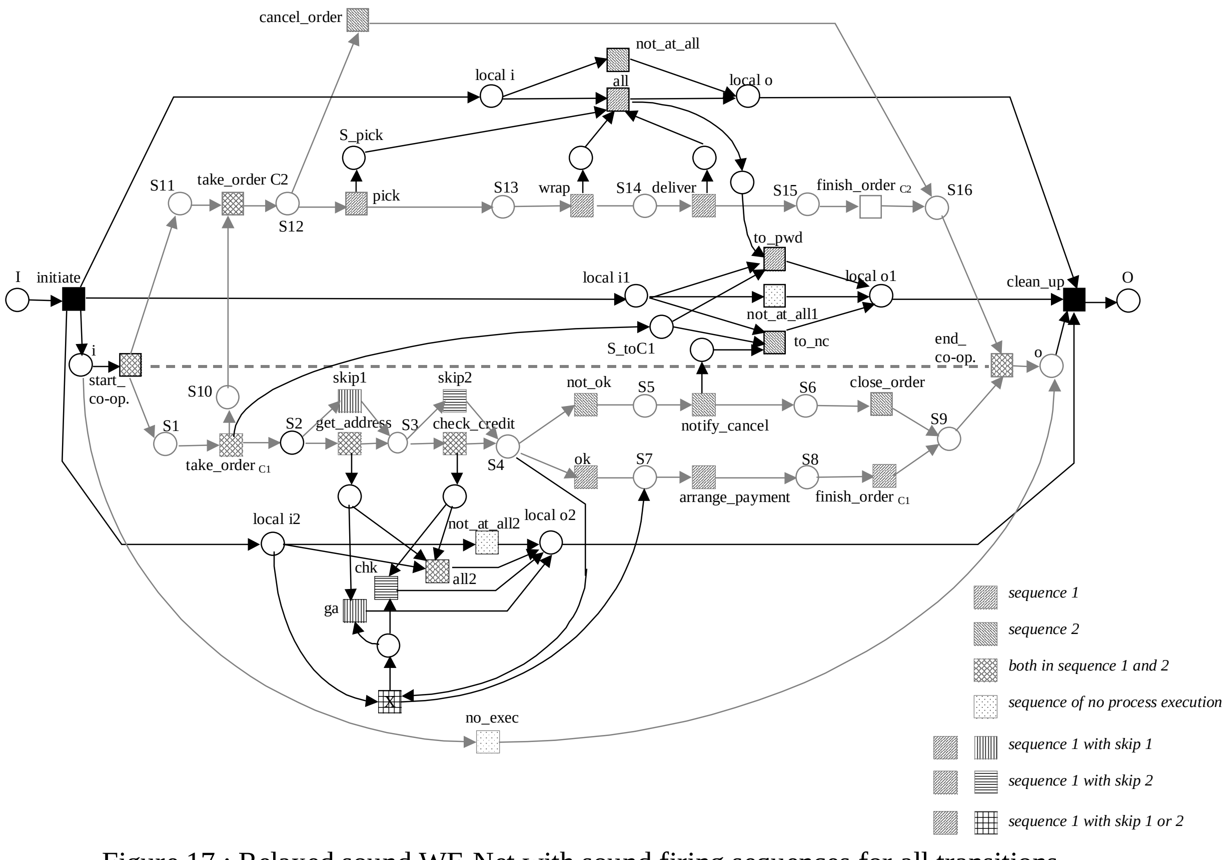


Figure 17 : Relaxed sound WF-Net with sound firing sequences for all transitions

7 Conclusions and future work

This paper introduces a new approach for specifying transaction management requirements for workflow applications. We propose to separate the specification of the workflow process and the transaction requirements. This allows the workflow and transaction designers to work independently. In addition, we specify transactional properties explicitly and independently from existing transaction models. This provides more flexible specification of transaction requirements.

Although our research distinguishes multiple transaction properties, we focus on atomicity in this paper. 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. In particular we think about relaxed sound Nets, that are only partially supported by transaction management facilities. In this case, the part without transaction management support would have to be transformed into a sound Net, while the part where advanced transaction support is offered, the relaxed soundness is sufficient and provides a more flexible and efficient way of execution.

Another direction for further research is to map the specified transaction properties to transaction models that are supported by commercial transaction management systems. This way we can delegate parts of the execution to these systems. Because of the explicit modeling of transaction requirements, we can also identify the properties that cannot be mapped onto existing transaction models. For those requirements we could either change the transaction specification or extend the transaction manager with the missing functionality.

Acknowledgements

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

References

Aalst97 Aalst, W.M.P. van der; Verification of Workflow Nets. In: Azema, P.; Balbo, G. (eds.); Application and Theory of Petri Nets 1997, LNCS 1248, Springer-Verlag, Berlin, 1997, pp. 407-426

Aalst98 Aalst, W.M.P. van der; Three good reasons for Using a Petri-net-based Workflow Management Systems, In: Wakay98, 1998, pp. 161-182

Aalst98b 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

Aalst00 Aalst, W.M..P. van der; Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In: Business Process Management: Models, Techniques, and Emperical Studies, LNCS 1806, Springer-Verlag, Berlin, 2000, pp. 161-183

AAEK+96 Alonso, G.; Agrawal, D.; El Abbadi, A.; Kamath, M.; Gunthor, R.; Mohan, C. Advanced transaction models in workflow contexts, ICDE96, New Orleans, 1996

AKAE+94 Alonso, G.; Kamath, M.; Agrawal, D.; El Abbadi, A.; Gunthor, R.; Mohan, C. Failure Handling in large scale workflow management systems, Technical Report RJ 9913 (87293), IBM Almaden Research Center, 1994

Alons97 Alonso, G. Processes + Transactions = Distributed Applications, Proceedings of the 7th international Workshop on High Performance Transaction Systems (HPTS'97), Asilomar, 1997

AAH98 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

ASSR93 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

BCSS00 Burkhard, H.D.; Czaja, L.; Skowron, A.; Starke, P. Workshop Concurrency, Specification & Programming. Informatik-Bericht 140, 2000

Chrys91 Chrysanthis, P.K.; ACTA, a framework for modeling and reasoning about extended transactions, PhD Thesis, Dept. of Computer and Information Science, University of Massachusetts, Amherst, 1991

CR94a Chrysanthis, P.; Ramamritham, K.; Synthesis of extended transaction models using ACTA, ACM Transactions on Database Systems 19 (3), 1994, pp. 450-491

Davie78 Davies, C.T. Data processing spheres of control, IBM Systems Journal 17(2), 1978, pp. 179-198

Elmag92 Elmagarmid, A. (ed.) Database Transaction Models for Advanced Applications, San Mateo, 1992

GH94 Georgakopoulos, D.; Hornick, M.F.; A Framework for Enforceable Specification of Extended Transaction Models and Transactional Workflows., International Journal of Intelligent and Cooperative Information Systems 3 (3), 1994, pp. 225-253

GHM96 Georgakopoulos, D.; Hornick, M.; Manola, F.; Customizing Transaction Models and Mechanisms in a Programmable Environment Supporting Reliable Workflow Automation, IEEE Transactions on Knowledge and Data Engineering 8 (4), 1996, pp. 630-649

GPS99 Grefen, G.; Pernici, B.; Sanchez, G. (eds.); Database support for Workflow Management - The WIDE Project, Kluwer Academic Publishers, Boston, 1999

GR93 Gray, J.; Reuter, A.; Transaction processing: concepts and techniques, Morgen Kaufmann Publishers Inc., San Francisco, 1993

HA98 Hagen, C.; Alonso, A. Flexible exception handling in the OPERA Process Support System, 18th Int Conf on Distributed Computing Systems (ICDCS98), Amsterdam, 1998

Hsu93 Hsu, M. (ed.), Special Issue on Workflow and Extended Transaction Systems 16, IEEE Computer Society, Washington, 1993

LG00 Ludwig, H.; Grefen, P.; Report on IDSO'00: The CAiSE*00 Workshop on "Infrastructures for Dynamic Business-to-Business Service Outsourcing", submitted for publication

JK97 Jojodia, S.; Kerschberg, L. (eds.); Advanced Transaction Models and Architectures, Kluwer, 1997

KMO98 Kiepuszewski, B.; Muhlberg, R.; Orlowska, M.; FlowBack: Providing Backward Recovery for Workflow Systems, Laura M. Haas, Ashutosh Tiwary (Eds.); Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA, ACM Press, 1998, pp. 555-557

Leyma95 Leymann, F.; Supporting Business transactions via partial backward recovery in workflow management systems, GI-Fachtagung Datenbanken in Buero Technik und Wissenschaft BTW'95, Springer Verlag, Dresden, 1995

LR00 Leymann, F.; Roller, D. Production Workflow: Concepts and Techniques, Prentice Hall, 2000

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

Mur89 Murata, T.; Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77 (4), April 1989, pp. 541-580

MAAE+95 Mohan, C.; Agrawal, D.; Alonso, G.; El Abbadi, A.; Guenthoer, R.; Kamath, M.; Exotica: A Project on Advanced Transaction Management and Workflow Systems, SIGOIS Bulletin (Special Issue on Business Process Management Systems: Concepts, Methods and Technology) 16 (1), 1995, pp. 45-50

Rei85 Reisig, W.; Petri Nets, EATCS Monographs on Theoretical Computer Science 4, Springer-Verlag, 1985

Reute89 Reuter, A.; ConTracts: A Means for Extending Control Beyond Transaction Boundaries, Proceedings of the 3rd International Workshop on High Performance Transaction Systems, Asilomar, 1989

Reute97 Reuter… in: JK97@@@

Roch00 Roch, S. Extended computation tree logic. In: BCSS00, 2000, pp. 225-234

SR93 Sheth, A.; Rusinkiewicz, M.; On transactional Workflows, In: Hsu93, 1993

TV95 Tang, J.; Veijalainen, J.; Transaction-oriented Workflow Concepts in Inter-organizational Environments, Proceedings of the 4th international Conference on Information and Knowledge Management , Baltimore, 1995, pp. 250-259

TV98 Tang, J.; Veijalainen, J.; Enforcement of Inter-Task Dependencies in Workflows, Characterization and Paradigm, International Journal of Cooperative Information Systems 7 (1), 1998, pp. 19

VA00 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

VDGK00 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

Wakay98 Wakayama, T. et al. (eds.); Information and Process Integration in Enterprises: Rethinking documents, Kluwer Academic Publishers, Norwell, 1998