

A Formal Foundation for Workflow Composition, Workflow View Definition, and Workflow Normalization based on Petri Nets

Victor Pankratius

Wolffried Stucky

AIFB Institute
University of Karlsruhe
76128 Karlsruhe, Germany
Email: {pankratius, stucky}@aifb.uni-karlsruhe.de

Abstract

In retail information systems it is common practice to subsume the data of products into product groups, which offers organizational advantages for example when new branches are opened, because they are assigned product groups instead of single products. Inspired by this approach, this paper focuses on business processes and proposes the usage of workflow modules stored in a workflow warehouse, which represent reusable, standardized components that are used to build more complex workflow models. In particular, the paper provides a formal foundation for such compositions in form of a workflow algebra based on Petri nets, which has similar operators known from relational algebra in databases. In addition, the need for a concept of workflow normalization is presented, which arises during the composition of workflow modules. Finally, it is shown how the algebra can also be applied in the context of Web service composition.

Keywords: Relational Data Model, Relational Algebra, Petri Nets, Workflow Composition, Workflow View Definition, Workflow Normalization, Petri Net Algebra, Web Services

1 Introduction

The multitude of available products that are to be managed by retail information systems (e.g., in supermarkets) has lead to the creation of so-called product groups, which represent sets of particular products (e.g., meat or dairy products). Product groups ease organizational tasks, especially when new branches are opened, because the products to be sold need not be chosen one by one. In addition, product groups represent a standardized range of products available in all branches, thus easing administrative and logistic tasks.

Inspired by this approach, this paper takes a closer look at business processes and workflows in such scenarios. Similar to product groups which are a set of products, we use the term *workflow modules* to denote sets of interrelated activities. We regard a workflow as the part of a business process in the real world which is automated by computers. Throughout the paper Petri nets will be used to model workflows (Petri 1962, Aalst 1996). In addition, this paper draws the attention to applications where predefined, standardized workflow models need to be combined to

create new workflow models. As a practical example, one might think of supermarkets or automotive manufacturers opening new branches which usually have very similar activities. To a large extent, workflow models for different branches may be identical, but often some minor modifications need to be done in order to account for the peculiarities of particular locations. In order to take advantage of economies of scale also on a modelling level by combining predefined workflow models, a precise notation is needed which can be used as a basis for composition languages. The focus of this paper are three related aspects of this composition problem: 1) to provide a formal notation for such compositions in form of a *workflow algebra* based on Petri nets, which allows to express the creation of a workflow model from other models using an algebraic notation with operators similar to those known from relational algebra in databases. 2) The paper also takes the opportunity to propose a definition of *workflow views* using the workflow algebra, which is more general than the definitions currently found in the literature (Avrilionis & Cunin 1995, Chiu, Cheung, Till, Karlapalem, Li & Kafeza 2004). 3) Continuing in the composition context, it is shown that the composition problem is not easily solved by a “copy & paste” approach, because a careless design of workflow models using compositions of smaller modules may introduce unwanted properties such as redundancies and other anomalies in the resulting model. To tackle this problem, the need for a notion of *workflow normal forms* is discussed.

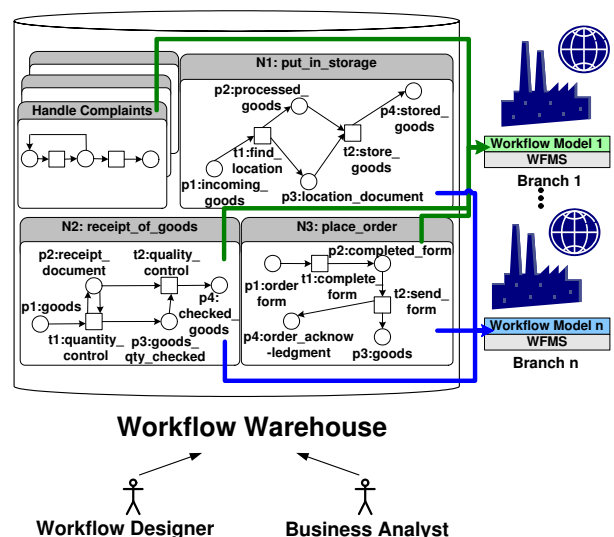


Figure 1: Workflow Composition Scenario.

Figure 1 shows a practical scenario in which a repository, which we call *workflow warehouse*, is used

to store predefined workflow modules. The workflow modules in such a warehouse are created and annotated with metadata by workflow designers. Business analysts verify and validate the models and make sure that they conform to business practices. The created workflow models represent general reference models for small problem domains (e.g., receipt of goods, handling of complaints, etc.). When a branch is opened, a specific workflow model has to be created that is tailored to the requirements of that branch. However, because of similar requirements of branches, some of the predefined workflow modules can be combined to form an initial model that is subsequently changed. After completion, the model is used in a local *Workflow Management System (WFMS)* which provides each participant with the right application, the right data, the right inputs at the right time (Georgakopoulos, Hornick & Sheth 1995).

Similar to our workflow warehouse, the notion of a *process warehouse* or *repository* already exists (Nishiyama 1999, Gruhn & Schneider 1998). The focus, however, is more on a general information source for software process improvement. In the workflow literature, *workflow patterns* have already been developed (Aalst, Hofstede, Kiepuszewski & Barros 2000). By contrast, such patterns are more abstract than our workflow modules, because they describe elementary workflow operations in a general way (e.g., sequence, synchronization, etc.). In our approach, workflow modules are targeted at particular problem domains and represent more complex workflows that are created for straight reuse. Based on workflow patterns, it has also been shown that Petri nets have advantages over *process algebras* (Best, Devillers & Koutny 1998) like π -calculus (Milner 1999, Aalst 2003). For example, Petri nets are state-based instead of event-based (important for the expressive power), they have formal semantics despite their graphical nature, and many analysis techniques are available (Aalst & Hofstede 2002). Although various transformations have already been defined for Petri nets (Baumgarten 1996, Desel & Oberweis 1996), the formalizations in general do not allow to express such transformations as precise equations or do not focus on analogies to the relational model. Other approaches like *Model Management* (Bernstein, Halevy & Pottinger 2000), *graph grammars* (Rozenberg 1997), or algebraic approaches of graph transformation (Corradini, Montanari, Rossi, Ehrig, Heckel & Löwe 1997) are too general to be used in the workflow problem domain right away. Nevertheless, our proposal is related to these approaches from a general point of view.

The paper is organized as follows. We recapitulate some terms of the relational data model in Section 2. Some basic definitions for Petri nets which are used throughout the paper are introduced in Section 3. A workflow algebra based on Petri nets is presented in Section 4. The applications of the algebra to view definitions on workflows are presented in Section 5. Section 6 introduces the notion of workflow normal forms. Some applications of the algebra to Web services are shown in Section 7. Finally, we discuss the results and mention some aspects for future work.

2 The Relational Data Model

In the *Relational Data Model* (Codd 1970), data in a database is organized in “tables”. Each “table” has a set of attributes $A = \{a_1 \dots a_n\}$ with domains $dom(a_i)$, and is described using the mathematical concept of a *relation* R which is a subset of the cartesian product of the attribute domains, i.e., $R \subseteq dom(a_1) \times \dots \times dom(a_n)$. A “row” r in a rela-

tion (other than the header row which contains the attribute names) is referred to as a *tuple*, denoted as $r \in R : r = (r_1, \dots, r_n)$. Conceptually, *relational algebra* is used to construct new relations that contain tuples satisfying some given properties, using other relations. This is done using a few basic operators like *selection* (σ , used to select a subset of tuples satisfying a given condition), *projection* (π , picks only specified attribute columns from a relation), *join* (\bowtie , used to combine related tuples from two relations into one single tuple), *union* (\cup , includes all tuples from two compatible relations into one relation without duplicates), and *set difference* (\setminus , used to include all tuples that are in one of two compatible relations, but not in the other). Furthermore, there are several types of join. For example, the *theta-join* is a join which allows to specify a join condition, whereas the *natural join* of two relations pairs only those tuples which agree in the attributes common to both relations (i.e., attributes that have the same name in both relations). Other operators, like for example, *intersection*, can be constructed by using the elementary operators mentioned before. Expressions of relational algebra can be applied to atomic operands (i.e., relations) or other expressions. Furthermore, all operands and results of expressions can be treated as sets. For further details and extensions we refer to (Garcia-Molina, Ullman & Widom 2002).

3 Petri nets

In analogy to database systems, we want to consider Petri nets as a “counterpart” to relations in the relational data model.

Definition 1 A Petri net $N = (P, T, F)$ is a directed bipartite graph consisting of a finite set of places P , a finite set of transitions T with $P \cap T = \emptyset$, $P \cup T \neq \emptyset$, and a flow relation F that represents a set of directed arcs $F \subseteq (P \times T) \cup (T \times P)$, going from places to transitions or transitions to places. \square

Definition 2 In a Petri net with n places, each place $p \in P$ may contain zero or more tokens. Let $x_i \geq 0$ denote the number of tokens assigned to a place p_i . A marking is an n -vector of the form $M_0 : (x_1, \dots, x_n)$ which shows the number of tokens assigned to each place p_i . A Petri net N with an initial marking M_0 is denoted (N, M_0) . \square

As stated above, it is not allowed to connect nodes of the same type. Furthermore, we assume each place or transition to have a name (e.g., “p1”) and a label (e.g., “goods”). However, names and labels will be omitted from the formalism, since they would introduce unnecessary complexity for the following argumentations. The set of places with a connection to a transition t are denoted $\bullet t$ and called *input places* (or pre-set) of t , whereas the set of places which are being connected starting from t via a directed arc are called *output places* (or post-set) of t , denoted $t\bullet$. The sets of transitions for a place p , $\bullet p$ and $p\bullet$, are defined accordingly. In a Petri net, places are passive, and transitions are active components which may change the state of places, according to *firing rules*. A transition is said to be *enabled*, if each of the input places contains at least one token. An enabled transition may fire, i.e., it decreases the number of tokens in all input places by one and increases the number of tokens in all output places by one. Transitions can be interpreted as *workflow tasks* that have to be performed. On the other hand, places can be interpreted as *repositories* which store tokens (i.e., objects) that are manipulated by tasks. If places contain one token, they can also be interpreted as *conditions* which evaluate to

true or false. Finally, tokens represent *artifacts* that have been created by tasks using some input. Sometimes tokens are also referred to as “cases”, because they may represent real-world entities like products or work assignments which progress through the workflow process (Aalst & Hee 2002). The term *workflow instance* is used to denote a copy of a workflow graph (i.e., the Petri net) created for execution, together with the associated cases.

Various extensions to Petri nets have been proposed, like for example higher-order Petri nets (Peterson 1977, Murata 1989, Aalst & Hee 2002), in which a place or transition can also be hierarchically refined using a subnet. We will give further details throughout the paper. Furthermore, the workflow models in our workflow repository represent reference models without associated cases, and we will only consider Petri nets without tokens in the following. Thus, the paper focuses on the analysis of the structure of workflows, not on their dynamic properties.

4 A Workflow Algebra

We now investigate the possibilities to modify and construct new Petri nets from given Petri nets. We introduce an approach which is inspired by operators known from relational algebra in databases. This approach should also show possible connections between the areas. Similar to relational algebra, we define the operators *selection*, *projection*, *join*, *union*, and *difference* for Petri nets. The proposed algebra is *closed* in that each of the operators applied to one or more Petri nets results again in a Petri net. The advantages of such an algebraic notation are that a series of compositions or other transformations of workflow models can be precisely written down using a single expression. Furthermore, the algebra could represent a basis for the development of programming languages for the composition of workflow models or Web services.

Throughout the paper, we will use the following general notations:

Given Petri nets $N = (P, T, F)$, $N' = (P', T', F')$

- a) $\pi_1(F) := \{x \mid \exists y : (x, y) \in F\}$
- b) $\pi_2(F) := \{y \mid \exists x : (x, y) \in F\}$
- c) $N' \subseteq N \Leftrightarrow N'$ is a subnet of $N \Leftrightarrow P' \subseteq P$, $T' \subseteq T$, $F' = F \cap ((S' \times T') \cup (T' \times S'))$

In a) and b), π_1 and π_2 are defined to choose the first and the second element of a tuple in F . In c), a shorthand notation for a subnet is defined.

Selection We interpret the flow relation F as a database relation which contains a set of pairs representing directed arcs. The arcs start in a place (transition) given by the first element of a tuple and end in a transition (place) specified by the second element of a tuple. In principle, we want the selection operator to select a set of directed arcs from a Petri net together with the places and transitions that they connect, and thus produce a new Petri net. More precisely:

Definition 3 (selection) Given a Petri net $N = (P, T, F)$ and a set of pairs S representing the arcs to be selected from N , we define the workflow selection operator $\sigma_S(N)$ as a transformation that produces from N a net $N' = (P', T', F')$ with

$$\begin{aligned} F' &:= F \cap S \\ P' &:= P \cap (\pi_1(F') \cup \pi_2(F')) \\ T' &:= T \cap (\pi_1(F') \cup \pi_2(F')) \end{aligned}$$

□

In F' , all directed arcs from F are included that are specified in the selection set S . π_1 and π_2 are used to

include the places and transitions that are connected by arcs in F' into P' and T' , respectively. Furthermore, we want to use the following shorthand notations given $p \in P$, $t_i \in T$, $p \bullet = \{t_1, \dots, t_n\}$ with all t_i different:

$$(p, *) = (p, t_1), (p, t_2), \dots, (p, t_n)$$

Accordingly, we use $(*, p)$, $(t, *)$ and $(*, t)$. For example, for the Petri net N in Fig. 2a), the expression

$$N' = \sigma_{\{(Task2, *), (*, Task2)\}}(N)$$

returns the highlighted net. Note that it is not allowed to select just one place or transition, since there is no workflow with just one element. We mention that there are other ways to extend the workflow selection operator (e.g., using path expressions, etc.). However, such extensions can be expressed using the definition presented above.

Difference The *workflow difference operator* (\setminus) can be used to “delete” a subnet N' from a given Petri net N , which results in a new net N'' , i.e.,

Definition 4 (difference) Given a Petri net $N = (P, T, F)$ and a subnet $N' \subseteq N$ with $N' = (P', T', F')$. The difference $N \setminus N'$ is a transformation that produces from N, N' a net $N'' = (P'', T'', F'')$ with

$$\begin{aligned} P'' &:= P \setminus P' \\ T'' &:= T \setminus T' \\ F'' &:= F \cap ((P'' \times T'') \cup (T'' \times P'')) \end{aligned}$$

□

In the definition above, the arcs going from the given net N to the subnet N' are also deleted. Figure 2b) shows an example for $N'' = N \setminus N'$ with N and N' taken from Fig. 2a).

Projection The idea behind projection in context of a workflow is to find a subnet of a Petri net which has only places or transitions on its “border” to the rest of the net, and substitute it by a single place or transition, respectively. Although similar maps are known in the Petri net literature (Reisig 1992, Baumgarten 1996), we want to keep the database perspective and the term “projection” for this operator. Thus, we introduce two projection operators which project a subnet either on a place or on a transition.

More precisely, we look at a Petri net $N = (P, T, F)$ and a subnet $N' \subseteq N$, with $N' = (P', T', F')$. Furthermore, the set B (standing for “border”) will denote the set of places and transitions of the subnet N' , which are connected with the rest of the net N , i.e.,

$$B(N', N) = \{x \in P' \cup T' \mid (x \bullet \cup \bullet x) \setminus (P' \cup T') \neq \emptyset\}$$

with $x \bullet$ and $\bullet x$ applied with regard to N . Furthermore, we will use the terms *place-bordered subnet* and *transition-bordered subnet* to denote a subnet whose set B contains only places or transitions, respectively.

Definition 5 (place projection) Given a Petri net N , a place-bordered subnet $N' \subseteq N$, and a place $p_{new} \notin P$ (used to substitute the subnet). We define the place projection operator $\pi_{P(B \rightarrow p_{new})}(N)$ as a transformation that produces from N, N' a net $N'' = (P'', T'', F'')$ with

$$\begin{aligned} P'' &:= (P \setminus P') \cup \{p_{new}\} \\ T'' &:= T \setminus T' \\ F'' &:= \Phi(F \setminus F'), \end{aligned}$$

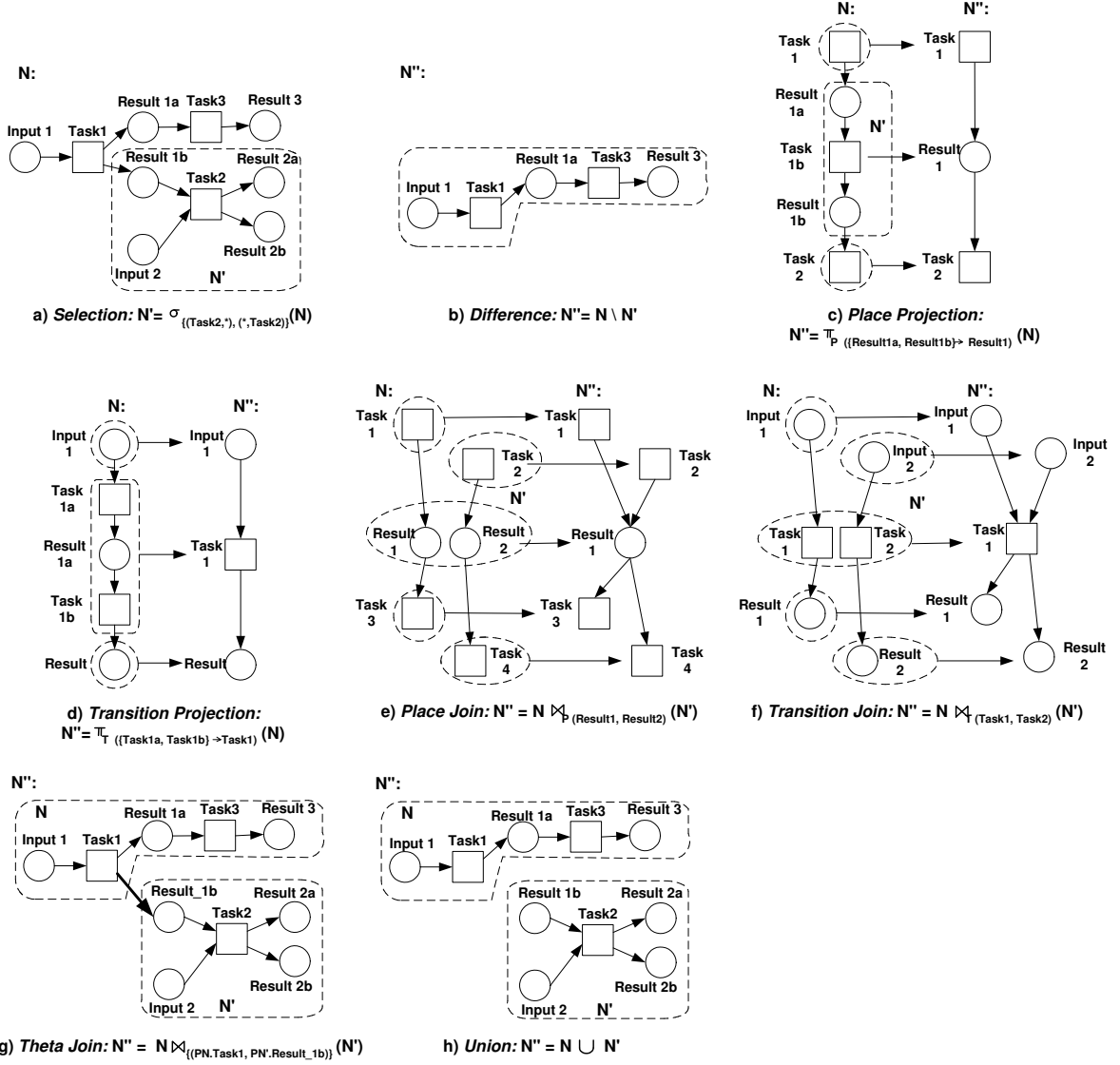


Figure 2: Operations from relational algebra defined for workflows which are represented as Petri nets.

$$\Phi(F) = \{\varphi(x, y) \mid (x, y) \in F\}$$

$$\varphi(x, y) = \begin{cases} (x, y) & \text{if } x, y \in (P \cup T) \setminus (P' \cup T') \\ (x, p_{new}) & \text{if } y \in B(N', N) \\ (p_{new}, y) & \text{if } x \in B(N', N) \end{cases}$$

□

In the definition above, the function φ is used to relocate arcs (which are coming from “outside” into the subnet or which go from the subnet to the “outside”) to the new place. Similarly, for a transition-bordered subnet $N' \subseteq N$, we define the *transition projection operator* $\pi_{T(B \rightarrow t_{new})}(N)$. For example, Fig. 2c) and 2d) show the results for the expressions:

$$N'' = \pi_P(\{Result1a, Result1b\} \rightarrow Result1)(N)$$

$$N'' = \pi_T(\{Task1a, Task1b\} \rightarrow Task1)(N)$$

The inverse operation of a projection – the substitution of a place or transition by a subnet with either only places or transitions on the border – is also a useful operation to hierarchically design workflow models. Such an operation is known as *refinement* in the Petri net literature (Reisig 1992). In our algebra, we can express the semantics of such an operator by using the difference and the join operator presented next.

Join In analogy to databases, we introduce two types of join to combine Petri nets: *natural join* (referred to as join in the following) and *theta join*.

Natural Join. Similar to the natural join operator in databases, the natural join operator for Petri nets can be used to create a new Petri net from two given Petri nets, which have both a place-bordered or transition-bordered subnet regarded as “common”. Because we have shown that the projection operator can be used to replace such subnets by places or transitions, we define the join operator for common places or transitions, and not for subnets (see Fig. 2). Again, two natural join operators are needed: one for joining on places (that will be called *place join*, denoted as \bowtie_P) and one for joining on transitions (called *transition join*, denoted as \bowtie_T).

Definition 6 (place join) Given two Petri nets $N = (P, T, F)$, $N' = (P', T', F')$ with $P \cap P' = \emptyset$, $T \cap T' = \emptyset$, a set of pairs $J = \{(p_1, p'_1) \dots (p_n, p'_n)\}$ with $p_i \in N$ and $p'_i \in N'$, which represent the places on which N and N' will be “glued” together. The place join $N \bowtie_P J(N')$ is a transformation that produces from N, N' a new Petri net $N'' = (P'', T'', F'')$ with

$$\begin{aligned}
\tau_2(J) &:= \{y | \exists x : (x, y) \in J\} \\
P'' &:= (P \cup P') \setminus \tau_2(J) \\
T'' &:= T \cup T' \\
F'' &:= \Phi(F \cup F')
\end{aligned}$$

$$\varphi(x, y) = \begin{cases} (x, y) & \text{if } x, y \in P \cup T \\ (x, y) & \text{if } x, y \in (P' \setminus \tau_2(J)) \cup T' \\ (x, p_i) & \text{if } y = p'_i \in \tau_2(J) \\ (p_i, y) & \text{if } x = p'_i \in \tau_2(J) \end{cases}$$

□

The two nets are joined together on the places specified in J . Then, the function φ is used to “relocate” arcs. The definition for the *transition join operator* $N \bowtie_{T, J} (N')$ is similar and we omit it here. Figure 2e) and 2f) depict examples for the expressions:

$$\begin{aligned}
N'' &= N \bowtie_{P\{(Result1, Result2)\}} (N') \text{ and} \\
N'' &= N \bowtie_{T\{(Task1, Task2)\}} (N').
\end{aligned}$$

The semantics behind a place join is that places of different nets are merged to more centralized repository of results. This type of join can be used to model mutual exclusion in workflows. In the Petri net community, such a composition is called *fusion* (Desel & Oberweis 1996). The semantics of a transition join operation is that tasks of different nets are merged together to a new task, which can only be performed if all inputs are available. Therefore, the transition join introduces synchronization. This is why the Petri net community refers to such transformations as *synchronization* (Desel & Oberweis 1996).

Theta Join. The theta join allows to combine Petri nets N and N' and add additional arcs between them, specified in a connection set. This way, other “branches of execution” described in N' can be added, which were previously not included in N .

Definition 7 (theta join) Given two Petri nets $N = (P, T, F)$ and $N' = (P', T', F')$ with $P \cap P' = \emptyset$, $T \cap T' = \emptyset$. The theta join $N \bowtie_{connection_set} (N')$ with $connection_set \neq \emptyset$ is a transformation that produces from N, N' a new net $N'' = (P'', T'', F'')$ with

$$\begin{aligned}
connection_set &\subseteq \{(P \times T') \cup (T \times P') \cup \\ &\quad \cup (P' \times T) \cup (T' \times P)\} \\
P'' &:= P \cup P' \\
T'' &:= T \cup T' \\
F'' &:= F \cup F' \cup connection_set
\end{aligned}$$

□

The theta join operator is able to connect transitions (places) from one net with places (transitions) of the other net and vice versa. An example for

$$N'' = N \bowtie_{\{(N.Task1, N'.Result1b)\}} (N')$$

is depicted in Fig. 2g).

Union In the context of workflows and Petri nets we regard a union of two Petri nets N and N' as the union of their places, transitions, and flow relations.

Definition 8 (union) Given two Petri nets $N = (P, T, F)$ and $N' = (P', T', F')$. The union $N \cup N'$ is a transformation that produces from N, N' a new net $N'' = (P'', T'', F'')$ with

$$\begin{aligned}
P'' &:= P \cup P' \\
T'' &:= T \cup T' \\
F'' &:= F \cup F'
\end{aligned}$$

□

The result of the union operator can express for example that N and N' are workflows which are executed concurrently. In contrast to relations in databases, the Petri nets do not have to be “compatible”. Figure 2h) shows an example for the expression:

$$N'' = N \cup N'.$$

Example As an example for the usage of the algebra, we assemble a custom workflow X using the modules shown in Fig. 1 inside the workflow warehouse. The result of the following expression is shown in Fig. 3:

$$\begin{aligned}
X &= N1 \bowtie_P \{(N1.p1, N2.p3)\} \\ &\quad (N3 \bowtie_P \{(N3.p4, N2.p2), (N3.p3, N2.p1)\} \\ &\quad (\sigma_{\{(p1, t1), (p2, t1), (t1, p3)\}}(N2)))
\end{aligned}$$

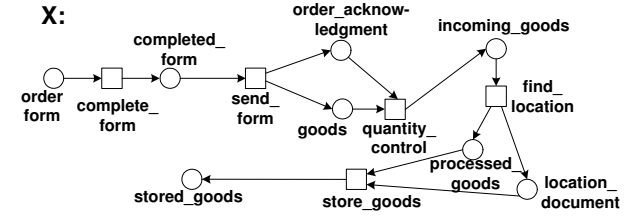


Figure 3: Example for a workflow composition using the workflow algebra.

First, a selection is made from the *receipt_of_goods* workflow (N2) without the *quality control*. Then, the *place_order* workflow (N3) is joined, and the places for *goods* (N3.p3 and N2.p1) and *order acknowledgement* (N3.p4 and N2.p2) are joined together, and the arcs are relocated. Finally, the *put_in_storage* workflow N1 is joined, and the places N1.p1 and N2.p3 are glued together.

Given Petri nets: $N = (P, T, F)$, $N' = (P', T', F')$, $N'' = (P'', T'', F'')$	
Properties:	
1)	$N \cup N' = N' \cup N$
2)	For $S_1 \subseteq F, S_2 \subseteq F, S_1 \subseteq S_2$: $\sigma_{S_1}(\sigma_{S_2}(N)) = \sigma_{S_1}(N)$
3)	For $S \subseteq (F \cup F')$: $\sigma_S(N \cup N') = \sigma_S(N) \cup \sigma_S(N')$
4)	For $S_1, S_2 \subseteq F$: $\sigma_{S_1 \cup S_2}(N) = \sigma_{S_1}(N) \cup \sigma_{S_2}(N)$
5)	For $N'' \subseteq N' \subseteq N$: $B'' = B(N'', N')$, $B' = B(N', N)$, and N'', N' place-bordered subnets: $\pi_P(B' \rightarrow p'_{new})(\pi_P(B'' \rightarrow p''_{new})(N)) = \pi_P(B' \rightarrow p'_{new})(N)$
6)	For $p2 \in P', \{t1 \dots tn\} = \bullet p2, \{t1' \dots tn'\} = p2 \bullet$: $N \bowtie_P \{(p1, p2)\} (N') = N \bowtie_{\{(t1, p1) \dots (tn, p1), (p1, t1') \dots (p1, tn')\}} (N' \setminus \sigma_{\{(p2, *)\}}(N')) \cup (N' \setminus \sigma_{\{(*, p2)\}}(N'))$

Table 1: Properties of the workflow algebra.

We finally point at some properties of the workflow algebra, presented in table 1, and give some brief ideas of the proofs. The commutativity of the union operator which follows from Def. 8 is expressed in (1). Equation (2) states that if a more special selection is made from a selection, the more special selection can be made right away, following from Def. 3. Equation (3) expresses that the selection from a union of two Petri nets can be split up, a result obtained from Def. 3 and 8. In (4) a selection with two unified selection sets can be replaced by two independent selections with one of the selection sets, followed by union of the results. This can also be obtained from Def. 3 and 8. In (5) it is stated that in a situation with two nested, place-bordered subnets, one projection can be omitted. Equation (6) shows that a (natural) place join between two nets N and N' on $p1$ and $p2$ can be expressed using the theta join, selection, difference, and union operators. The idea is to first select all parts of N' except the place $p2$ and except the arcs to and from that place, using the expression $(N' \setminus \sigma_{\{(p2,*)\}}(N')) \cup (N' \setminus \sigma_{\{(*,p2)\}}(N'))$. Then, using the theta join, the arcs from the transitions of N' pointing to $p2$ are relocated to $p1$; the same is done for the arcs going to the post-set of $p2$. Despite this realization we do not remove the place or transition join operators from the algebra, because they are used frequently. The properties (5) and (6) are similar for transition projection and transition join. Interestingly, we want to mention that if the notation was adapted to the relational data model and relational algebra (e.g., N and N' as relations; S, S_1, S_2 as conditions), the properties (1), (3), (4), (5) would be similar for relations. The natural join in the relational model can also be expressed using theta join and an additional projection (Elmasri & Navathe 2004). Of course, for a more thorough comparison, further adaptations are needed with respect to the exact notation and the peculiarities of the relational model (e.g., “compatible” relations for union, etc.), which we do not discuss here in more detail. In addition, there are also significant differences to the relational case, as other properties known from relational algebra do not hold in our case. For example, the selection operator has other properties (2). Thus, although the models have many commonalities, it cannot be said that there is an analogy in all respects. Nevertheless, such a close relationship between the models provides new ground for investigations if other concepts from the relational model (e.g., views, normalization) can be applied to workflows and Petri nets.

5 Workflow Views

Originating in the area of databases, *views* are virtual relations which are derived from existing relations or other views, for example by using relational algebra operators (Elmasri & Navathe 2004). Views are not materialized, and they are typically computed every time a user accesses them. A database user cannot distinguish between a relation or a view. However, data manipulation and update operations on views are restricted, because there may be ambiguities when such operations are mapped back to the original relations. A view definition is useful when the result of several joins of relations is used frequently, or when access to data has to be restricted, so that certain employees should see only some parts of the data.

Starting from database views, the need for a view concept on workflows has already been recognized in the literature (Avrillionis & Cunin 1995, Chiu et al. 2004). However, a *workflow view* is often regarded only as a sub-model of the original workflow

model, thus not directly corresponding to the definition of database views which allow the combination of several relations or views for the definition of a new view, using relational algebra. We therefore propose the usage of our workflow algebra for a more general definition of workflow views. We propose the usage of any expression of our algebra for the definition of a view on workflow models. Such a workflow view is not materialized and will be recomputed every time it is accessed.

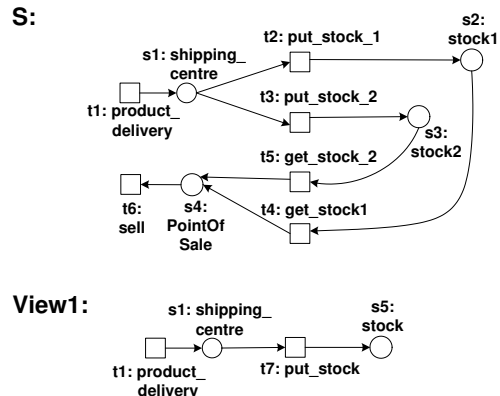


Figure 4: Example for a workflow view.

As a simple example, let us consider a practical scenario where a manager wants to streamline business processes to make them more efficient. For a discussion with the board of directors, he needs to know the main activities and their interconnections without too many details, based on the workflow model used in the company (e.g., S in Fig. 4), starting from product delivery to storage (he will unexpectedly ask the employees to do this quickly, like all managers do). To solve this problem, *View1* can be defined using the expression:

$$View1 = \sigma_{\{(t1,*), (s1,*), (*,s5)\}} \left(\pi_{\{t2,t3 \rightarrow t7\}} \left(\pi_{\{s2,s3 \rightarrow s5\}} (S) \right) \right)$$

The view aggregates the places labelled *stock1*, *stock2* and the transitions labelled *put_stock1*, *put_stock2* from S , and selects only the desired part of the workflow. It should be noted that the “*” in the selection set has a dynamic aspect; if the underlying model was changed in the meantime, e.g., a new place was added in the post-set of $t1$, it would appear in the view. Thus, if the workflow models are changed frequently due to the adaptation to new requirements, the view always provides up-to-date, customized information. In addition, business analysts often differentiate between core processes which contribute to the core competencies of a company and its competitive advantage, and support processes which provide support for the core business processes. Using views, the current core processes and support processes can be extracted from the company-wide workflow model. Other operations also make sense when applied on workflow views. A union may combine different pieces of workflows or other views to one big overview for the manager. This is very useful to supervise the activities of several projects that run in parallel. Difference may be used to cut execution paths which are not in the focus of the analysis. Joins can be used to present a simplified model of *interorganizational workflows*. For example, a view for company A shows a simplified model of a manufacturing process in which products are output to a place A_O . Another view for company B in the same supply chain uses the products of company A as input to its production process, modelled by a place B_I . The general “big

picture” would be obtained by joining the two views on A_O and B_I . Such an approach can be very useful in *Supply Chain Management (SCM)* to analyze and streamline processes which are crossing several companies. Furthermore, during *mergers & acquisitions* of companies, a workflow analysis based on views can also help to find interfaces and integrate the business processes of an acquired company more efficiently.

Just like in the database area where operations on views need to be restricted, in workflow views not all combinations of workflows may make sense with respect to the underlying workflow. Since this depends on the specific situation and workflow, such constraints need to be verified by a runtime system. In addition, we mention that workflow views also need to restrict the way tokens are shown in a view, because the firing rules in a view may not conform to the underlying workflows. For example, this may occur in situations where several places are projected onto one single place. However, a solution to this problem is out of scope here, since the focus is on the structure of the workflows (Petri nets without tokens), not the dynamic aspects. Nevertheless, it would be advantageous to show tokens in a view as progress and status indicators.

6 Workflow Normalization

Continuing with the composition scenario shown in Fig. 1 we argue in this section that a notion of normal forms for workflows is needed. As with database relations, a careless design of workflow models using Petri nets can lead to problems. For example, a workflow designer may be tempted to use “copy & paste” functionality of an editing tool (based on our workflow algebra) to copy a module from the workflow warehouse (cf. Fig. 1) and paste it multiple times at different locations in the workflow model he or she is designing for a branch. This introduces *anomalies*, some of which are similar to those found in relational databases (Elmasri & Navathe 2004):

1. **Redundancy.** The information contained in the workflow modules is repeated unnecessarily in the workflow model.
2. **Update Anomalies.** Suppose that a workflow module named “place order” was inserted multiple times at different locations in the workflow model. If suddenly an extension has to be made (e.g., a transition has to be added), the same insertion has to be made in multiple locations (we assume that this change is only needed in that particular workflow model, so that it is not appropriate to modify the global module in the workflow warehouse). If possible, it would be much better to have just one copy of the module in the whole workflow model and thus, only one insertion.
3. **Inconsistent Behaviour.** In the example above, the workflow designer may forget to make the changes to all copies of the “place order” module in the model. Such inconsistent changes lead to different, *unintended workflow behaviour* which does not conform to the specifications.

The anomalies presented above are definitely undesirable and should be prevented by using a systematic conceptual approach. Although the methodology and the properties may slightly differ in the following from databases, we chose to stick to the term “normalization”, because in our view, the same goal is pursued: a better design that avoids anomalies. In databases, so-called *functional dependencies*

are analyzed to eliminate some anomalies (Elmasri & Navathe 2004). In our workflow context, we analyze the *flow dependencies* along with the structure of a Petri net. As in the rest of the paper, we stick to Petri nets without tokens. In analogy to databases, we propose the notion of *workflow normal forms* for Petri-net-based workflow models. The main intention is to specify criteria for a “good” structure of workflows. Such criteria are also especially useful in business process reengineering projects, in which existing workflow models need to be analyzed and restructured. We point out that our notion of workflow normal forms should not be confused with *normalized Petri nets* found in the Petri net literature (Tiplea & Tiplea 1998), which denote Petri nets whose weight function and initial and final markings take only values in $\{0, 1\}$.

Definition 9 (1NF) We define a workflow represented as a Petri net to be in First Normal Form (1NF), if all places or transitions are atomic, i.e. if no place or transition is hierarchically decomposable into sub-workflows. \square

The 1NF is intended to flatten out hidden sub-workflow specifications of a workflow. This is necessary for the identification of redundant flow specifications, and to reveal the real structure of the net. Based on a workflow model in 1NF, further analysis can be done. For this, we first need to introduce some other definitions:

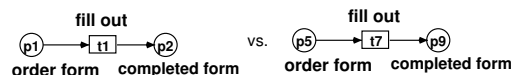
Definition 10 (Petri net morphism) Given Petri nets $N = (P, T, F)$ and $N' = (P', T', F')$. A structure-respecting Petri net morphism is a map $\varphi : P \times T \rightarrow P' \times T'$ in which for every $(x, y) \in F$ holds: 1) If $(x, y) \in F \cap (P \times T)$ then either $(\varphi(x), \varphi(y)) \in F' \cap (P' \times T')$ or $\varphi(x) = \varphi(y)$ and 2) If $(x, y) \in F \cap (T \times P)$ then either $(\varphi(x), \varphi(y)) \in F' \cap (T' \times P')$ or $\varphi(x) = \varphi(y)$.

Definition 11 (isomorphism) A bijective map $\rho : P \cup T \rightarrow P' \cup T'$ that preserves the structure of places and transitions is called an isomorphism, i.e., the map of the net $N = (P, T, F)$ onto $N' = (P', T', F')$, is an isomorphism if $\forall x, y \in P \cup T : (x, y) \in F \Leftrightarrow (\rho(x), \rho(y)) \in F'$. Two nets are called isomorphic if there is an isomorphism from one net onto the other. \square

In general, Def. 10 provides a mapping from a source net to a target net in which some structural properties of the source net are preserved. The term “morphism” is generally used for the mapping as well as for the resulting net. Definition 11 (which is also a morphism) can be used to find identically structured subnets inside a workflow model.

Definition 12 (2NF) A workflow represented as a Petri net in 1NF is in second normal form (2NF) if there are no multiple occurrences of isomorphic subnets with identically labelled places or transitions. \square

In Sect. 3 we assumed that a place or transition can have a name (e.g., “p1”) and a label (e.g., “order form”). Definition 12 demands that there should be no subnets with identical structure *and* identical labels (even though the names of labels or transitions may be different, e.g.



). If a workflow model N is not in 2NF, there must be multiple copies (e.g., N'_1, N'_2) of a subnet N' at different

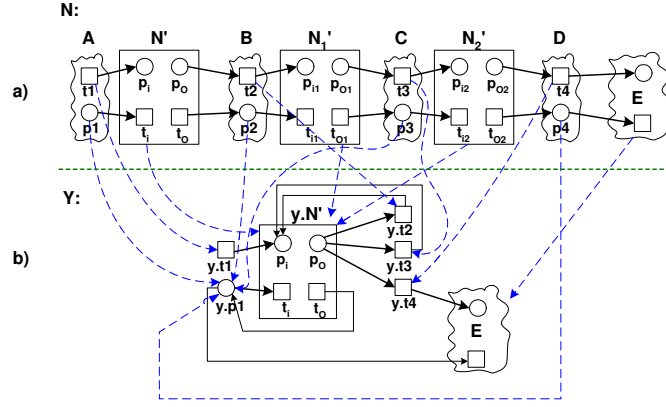


Figure 5: Example for normalization.

locations in the model N . This situation is suspect, since it points to conceptual errors which can be of two types: 1) N'_1, N'_2 are actually meant to be the same as N' . In this case, the workflow model has to be restructured to contain N' only once. 2) N'_1, N'_2 may occur in another context, and therefore be different, which justifies their existence. However, in this case the labels have to be renamed properly to reflect this different context (e.g., place labels “completed form” to “completed order form” and “completed goods receipt form”, respectively – reflecting different forms). Both actions lead to a workflow in 2NF.

Let us consider the case 1) where a workflow N needs to be restructured, depicted in general in Fig. 5a). In the figure, multiple copies (N'_1, N'_2) of the subnet N' occur multiple times in the workflow model (we assume that places and transitions have different names, but identical labels). Since the exact structure of N' is not of interest, no arcs are shown inside. The borders (see Sect. 4) connecting the copies of N' with other parts of the net consist of input and output places and transitions $\{p_i, p_o, t_i, t_o\}$. Each of these places and transitions represent a projection of a more complex subnet. The copies of N' are connected to other nets A, B, C, D , whose representations are also simplified assuming their border contains places as well as transitions, which are projected on one place and transition only. To avoid anomalies (e.g., when N' is updated) the structure of the workflow model has to be changed to contain N' only once. In principle, this can be accomplished using structure-respecting morphisms. An example for such a morphism is the folding operation:

Definition 13 (folding) *Given two Petri nets $N = (P, T, F)$ and $N' = (P', T', F')$. The folding operation is a surjective map $\psi : P \cup T \rightarrow P' \cup T'$, where*

$$\begin{aligned} \psi(P) &\subseteq P', \psi(T) \subseteq T', \text{ and} \\ (x, y) \in F &\Rightarrow (\psi(x), \psi(y)) \in F' \end{aligned}$$

□

The folding operation maps places from a net N to places of N' and transitions from N to transitions of N' and preserves the flow dependencies. The operation is useful to “fold” identically structured parts of a workflow model on top of each other and thus reduce their multiple occurrence (redundancy). In the Petri net literature (Baumgarten 1996), folding is compared to the creation of procedures and procedure calls in programming languages. As an example, Fig. 5b) shows how the folding operation can be applied to obtain a new workflow model with N'

occurring only once. Each place and transition of the net N' is mapped to those of $y.N'$; the places $p_1 \dots p_4$ are mapped onto $y.p_1$ that now represents an input/output repository for $y.N'$. The mapping of the transitions $t_1 \dots t_4$ has to be done with care. They cannot be mapped altogether on $y.t_1$, since they may represent *different* tasks outside N' . In addition, after mapping $t_2 \dots t_4$ onto $y.t_2 \dots y.t_4$, the firing order $t_2 < t_3 < t_4$, which represents a sequential processing, is lost. The firing order can be preserved by adding constraints, for example by using predicates or other constructs like inhibitor arcs (Peterson 1977).

In practice, several isomorphic subnets with identically labelled places or transitions may occur at different levels of aggregation. In such cases, the largest isomorphic subnets occurring multiple times should be identified first (e.g., using clustering (Keller 2003) or other algorithms (Kuramochi & Karypis 2004)) and the discussed procedure should be applied. The same can be done recursively with the subnet, until the 2NF of the workflow model is reached.

We point out that a “good” structure of a workflow model may also depend on the semantics related to labels of places and transitions, which cannot be inferred only by analyzing the structure. Places or transitions with different labels may *mean* the same. For example, two transitions performing the same task may be labelled “process request” and “work on customer inquiry”.

Definition 14 (3NF) *A workflow represented as a Petri net in 2NF is in third normal form (3NF) if there are no multiple occurrences of isomorphic subnets with semantically identical labels for places or transitions.* □

In our view, a workflow designer can be provided with some semi-automatic assistance for detecting such redundancies of places or transitions with different labels meaning the same, for example by using synonym dictionaries and concepts developed for the Semantic Web (Berners-Lee, Hendler & Lassila 2001). This implies additional effort during the design of a workflow module, in which places and transitions may be associated with concepts contained in an ontology (Staab & Studer 2003) that is valid for the whole warehouse. The ontology represents a taxonomy of classes of objects and relations among them – an information that can be used by an inference mechanism to automatically draw conclusions and make propositions to the workflow designer.

Finally, we mention that there are also other types of Petri net morphisms, each of which preserve different properties of a Petri net (see (Mikolajczak & Wang 2003) for a comparison), and which also take

tokens and the behaviour of nets into account. Based on different types of morphisms and the dynamic behaviour of Petri nets, further workflow normal forms can be developed. This may be especially beneficial for avoiding situations with deadlocks which may result from the composition process. Behaviour-based workflow normal forms may therefore represent further milestones for quality criteria during workflow composition. Since we concentrate on Petri nets without marking at this time, such an examination is beyond the scope of this paper.

7 Application to Web Services

In this section, we show how the developed concepts of the workflow algebra and workflow views can also be applied in the context of Web services. Web services play an important role in the workflow area, since they are used to implement activities of business processes (Leymann, Roller & Schmidt 2002).

Web services are independent software components that use the Internet as a communication and composition infrastructure. Building on a standardized stack of protocols, Web services provide a service-oriented view and abstract from specific computers (see (Casati & Dayal 2002) for details). Web service composition is used to create higher-level services based on other Web services, e.g., a travel agent may compose a hotel reservation and airline ticket service to offer a new service. In this context, the concept of *orchestration* is targeted at the interaction between Web services at message level and focuses for example on the execution order of interactions. *Choreography* refers to the overall message exchange between all Web services that participate in a collaborative exchange, and is for example directed towards exception handling or transactions. The strict separation between orchestration and choreography is blurring as various languages and standards have been developed: *Business Process Execution Language for Web Services* (BPEL4WS), *Web Services Choreography Interface* (WSCI), *Business Process Management Language* (BPML), XLANG, *Web Services Flow Language* (WSFL). A detailed comparison is given in (Peltz 2003, Aalst, Dumas & Hofstede 2003).

Using the workflow algebra introduced in Sec. 4, we can describe the creation and composition of services on a conceptual level. We assume that the behaviour of a Web service is described by an associated Petri net, which has one input and one output place. Unlike other Petri-net-based composition approaches (Hamadi & Benatallah 2003), we use the algebra not only to join input and output places of available services, but also to construct services from modules in the workflow warehouse (cf. Fig. 1). In this context, we assume that the warehouse contains predefined modules for business processes as well as modules needed for the technical realization of Web services, described as Petri nets.

A current problem in the area of Web service composition is that often the exact mode of operation and the “inside” of a Web service used for composition is not known. In many cases, the providers don’t want to reveal such details because of competitive advantages. Such information, however, could be important if a certain quality of service level has to be ensured in a composite service, or to avoid deadlocks with services from other providers. We propose the application of workflow views (cf. Sec. 5) as a compromise, which are derived from the Petri net associated with the service. When a developer requests the Petri net description associated with a service, a *view* on that Petri net is returned. Using the selection, projection, or difference operators (cf. Sec. 4), the parts that

the provider does not want to reveal to the public may be eliminated. In a more sophisticated scenario, different views with different levels of detail may be computed for different types of (paying) customers. The advantage is that the underlying functionality of the service and the Petri net model are not changed at all, since the view is computed dynamically upon request.

8 Conclusion

This paper addresses problems encountered during the composition of workflow modules. The need for theoretical constructs is motivated by the descriptions of practical scenarios from supermarket chains or automotive manufacturers which use a workflow warehouse to build new workflow models from existing ones. A formal notation (workflow algebra) for the composition of Petri nets is proposed which is inspired by operators known from relational algebra. To a certain extent, the properties of the workflow algebra are similar to those of relational algebra, and interesting relationships are revealed. We expect that users will adapt quickly to languages based on the proposed workflow algebra, due to the popularity of database concepts. Furthermore, using the workflow algebra, the concept of workflow views is extended in such a way that dynamically computed views on workflows may aggregate, eliminate, or combine parts of workflows. In addition, it is shown that a careless modelling and composition of workflow models may lead to unwanted anomalies – some of them similar to databases. Therefore, the need for a notion of workflow normal forms is discussed to tackle this problem. Finally, it is shown that the presented concepts can also be used to describe the creation and composition of Web services which are often used to implement activities of business processes. In this context, workflow views seem to be a promising solution for the delivery of the the internal processing model of a Web service, since they may aggregate or eliminate parts that should not be revealed to the public.

This paper also tries to identify potential future directions for workflow and business process management research, in particular with respect to workflow normalization algorithms. In the future, further investigations are needed to connect the results from Petri nets without tokens with Petri nets which contain tokens. In addition, the resulting implications for workflow views and workflow normal forms with respect to dynamic net behaviour have to be analyzed in more detail.

Acknowledgements. The first author would like to thank Phil Bernstein for the fruitful discussion at SIGMOD/PODS 2004 in Paris, France.

References

- Aalst, W. (1996), Three good reasons for using a petri-net-based workflow management system, *in* ‘Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC’96)’, pp. 179–201.
- Aalst, W. (2003), ‘Pi calculus versus petri nets: Let us eat humble pie rather than further inflate the pi hype’, <http://tmitwww.tm.tue.nl/research/patterns/download/pi-hype.pdf> (unpublished discussion paper) .
- Aalst, W., Dumas, M. & Hofstede, A. (2003), Web service composition languages: Old wine in

- new bottles?, in '29th EUROMICRO Conference 2003, New Waves in System Architecture, 3-5 September 2003, Belek-Antalya, Turkey', IEEE Computer Society, pp. 298–307.
- Aalst, W. & Hee, K. (2002), *Workflow Management. Models, Methods, and Systems*, MIT Press.
- Aalst, W. & Hofstede, A. (2002), Workflow patterns: On the expressive power of (petri-net-based) workflow languages., in K. Jensen, ed., 'Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)', Vol. 560 of *DAIMI*, University of Aarhus, Aarhus, Denmark, pp. 1–20.
- Aalst, W., Hofstede, A., Kiepuszewski, B. & Barros, A. (2000), Advanced workflow patterns, in O. Etzion & P. Scheuermann, eds, 'Proc. 7th International Conference on Cooperative Information Systems (CoopIS)', Vol. 1901 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 18–29.
- Avrilionis, D. & Cunin, P. (1995), Using views to maintain petri-net-based process models, in 'Proc. International Conference on Software Maintenance', pp. 318–326.
- Baumgarten, B. (1996), *Petri-Netze. Grundlagen und Anwendungen (in German)*, 4th edn, Spektrum Akademischer Verlag.
- Berners-Lee, T., Hendler, J. & Lassila, O. (2001), 'The semantic web', *Scientific American*.
- Bernstein, P. A., Halevy, A. Y. & Pottinger, R. (2000), 'A vision of management of complex models.', *SIGMOD Record* **29**(4), 55–63.
- Best, E., Devillers, R. & Koutny, M. (1998), Petri nets, process algebras and concurrent programming languages, Vol. 1492 of *Lecture Notes in Computer Science (LNCS)*, Springer, Berlin.
- Casati, F. & Dayal, U., eds (2002), *Special Issue on Web Services*, Vol. 25, IEEE Bulletin of the Technical Committee on Data Engineering.
- Chiu, D. K., Cheung, S., Till, S., Karlapalem, K., Li, Q. & Kafenza, E. (2004), 'Workflow view driven cross-organizational interoperability in a web service environment', *Information Technology and Management* **5**(3-4), 221–250.
- Codd, E. F. (1970), 'A relational model of data for large shared data banks', *Commun. ACM* **13**(6), 377–387.
- Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R. & Löwe, M. (1997), *Algebraic Approaches to Graph Transformation*, Vol. 1 of Rozenberg (1997), chapter 3, pp. 163–245.
- Desel, J. & Oberweis, A. (1996), 'Petri nets in applied computer science (in German)', *Wirtschaftsinformatik* **38**(4), 359–367.
- Elmasri, R. & Navathe, S. B. (2004), *Fundamentals of Database Systems*, 4th edition edn, Pearson Addison Wesley.
- Garcia-Molina, H., Ullman, J. D. & Widom, J. (2002), *Database Systems. The Complete Book.*, Prentice Hall, Inc.
- Georgakopoulos, D., Hornick, M. & Sheth, A. (1995), An overview of workflow management - from process modeling to workflow automation infrastructure, in 'Distributed and Parallel Databases', Vol. 3, pp. 119–153.
- Gruhn, V. & Schneider, M. (1998), Workflow management based on process model repositories, in 'Proceedings of the 20th international conference on Software engineering', IEEE Computer Society, pp. 379–388.
- Hamadi, R. & Benatallah, B. (2003), A petri net-based model for web service composition, in 'Proceedings of the Fourteenth Australasian database conference on Database technologies 2003', Australian Computer Society, Inc., pp. 191–200.
- Keller, W. (2003), 'Clustering for petri nets', *Theor. Comput. Sci.* **308**(1-3), 145–197.
- Kuramochi, M. & Karypis, G. (2004), 'An efficient algorithm for discovering frequent subgraphs', *IEEE Transactions on Knowledge and Data Engineering* **16**(9), 1038–1051.
- Leymann, F., Roller, D. & Schmidt, M.-T. (2002), 'Web services and business process management.', *IBM Systems Journal* **41**(2), 198–211.
- Mikolajczak, B. & Wang, Z. (2003), Conceptual modeling of concurrent systems through stepwise abstraction and refinement using petri net morphisms., in I.-Y. Song, S. W. Liddle, T. W. Ling & P. Scheuermann, eds, 'Proc. 22nd International Conference on Conceptual Modeling', Springer, Chicago, IL, USA, pp. 433–445.
- Milner, R. (1999), *Communicating and Mobile Systems: The π -Calculus*, Cambridge University Press, Cambridge, UK.
- Murata, T. (1989), Petri nets: Properties, analysis and applications, in 'Proc. IEEE', Vol. 77, pp. 541–580.
- Nishiyama, T. (1999), Using a "process warehouse" concept. a practical method for successful technology transfer., in '2nd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '99), May 2-5, 1999, Saint Malo, France', IEEE Computer Society, pp. 117–120.
- Peltz, C. (2003), Web services orchestration. a review of emerging technologies, tools, and standards., Technical report, Hewlett-Packard Company.
- Peterson, J. L. (1977), 'Petri nets', *ACM Comput. Surv.* **9**(3), 223–252.
- Petri, C. A. (1962), 'Kommunikation mit Automaten (in German)', *Schriften des Instituts für instrumentelle Mathematik*, University of Bonn, Germany.
- Reisig, W. (1992), *A Primer in Petri Net Design*, Springer Verlag.
- Rozenberg, G., ed. (1997), *Handbook of Graph Grammars and Computing by Graph Transformation*, Vol. 1, World Scientific.
- Staab, S. & Studer, R., eds (2003), *Handbook on Ontologies*, Springer, Berlin.
- Tiplea, F. & Tiplea, A. (1998), On normalization of petri nets, in 'Proc. of the 11th Romanian Symposium on Computer Science ROSYCS'98', Iasi, Romania.