Institut fuer Informatik
Humboldt-Universitaet zu Berlin
Mai 1998

Database Theory – Petri Net Theory – Workflow Theory

Ekkart Kindler*

Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, D-10099 Berlin†

1 Introduction

Workflow management is in vogue. Jablonski [Jab95] calls *workflow management* the *relational databases* of the 90ties. However, Jablonski states one important difference between workflow management in the 90ties and relational databases in the 80ties: Whereas there has been a theoretical foundation for relational databases right from the beginning (in the early 70ties), there is no well-established *workflow theory* yet.
In this paper we will argue that there is a *workflow theory*, which only needs to be rediscovered underneath a vast amount of existing theory. For example, some classical results from *Petri net theory* just need a new interpretation in order to make up a workflow theory. Van der Aalst [Aal97b, Aal97a, Aal98] has shown that simple Petri net concepts such as *liveness* and *boundedness* can be used to characterize *sound* workflows. We do not claim that this approach is a fully-fledged workflow theory yet; but we claim that a helpful workflow theory will be akin to it.
This paper will substantiate the above statement by providing a flavour of a future workflow theory. From a theoreticians point of view this theory may appear trivial because there is not much hope for deep theorems. But, we make a plea by forming an analogy to a well-established part of *database theory*: *normal forms* and *dependency theory* (cf. [Cod90, Dat90]). Dependency theory is even simpler and has even less deep theorems — but is a great success because it is just the bit of theory needed!
This paper is not meant to develop new ideas; we mainly present those of van der Aalst [Aal97a] and give a new interpretation to a result of Valette [Val79] (also rephrased in [Aal98]). The main purpose is to point out the right ideas from Petri net theory in the field of workflow management — and to direct research into this field.

2 Database theory

In order to form the analogy to database theory, let us briefly recall *normal forms* and *dependency theory* which have been a great success within the field of relational database systems [Dat90]. This theory was established by Codd in the early 70ties. Codd rephrases the main motivation for his theory as follows: ‘My main goal was to develop some theory that would be applicable to logical database design, and especially to the creation of a sound collection of base relations’ [Cod90], p. 317.
Normal forms capture formally, what a skilled database designer considers as a good design. Each normal form is motivated by *anomalies* which may occur in a bad database design. Let us consider an example of a simple database which consists of a single relation. The relation

*Supported by the Deutsche Forschungsgemeinschaft within the research project ‘Petri Net Technology’ (DFG Forschergruppe ‘Petrinetz-Technologie’).
†e-mail: kindler@informatik.hu-berlin.de

project administration consists of the following *attributes*: project, person, percentage of working hours spent on this project, and the telephone number of the person. Projects and persons are represented by some number. Essentially, a relation (or an instance of a relation) can be represented as a table as shown in Fig. 1 (without identical rows). The columns of the table correspond to the attributes of the relation.

PROJECT#	PERSON#	PERCENT.	TEL.
1	4711	15	208
2	4711	10	208
1	4712	100	209
1	4713	100	209
0	4711	30	208
2	4714	100	210
3	4715	100	207
0	4710	30	220
1	4710	10	220
	⋮		

Figure 1: The relation *project administration*

The relation *project administration* is not well designed. This will be demonstrated by means of some anomalies (insert and update). Suppose, there is a new person who is not yet associated with a particular project. Then, it is impossible to *insert* this person into this relation. The *insert anomaly* means that we cannot store the telephone number of a person within relation *project administration* unless this person works on a project. Next suppose, we want to change the telephone number of person `4711`; then we actually need to *update* three rows. The *update anomaly* means that updating one fact needs changes in several rows.
Now, why is the above relation not well designed? The reason is that attribute `TEL.` is *functionally dependent* (i.e. is fully determined by the value) of attribute `PERSON#`, since each person has only one telephone number. Another attribute `PERCENT.`, however, is not functionally dependent on attribute `PERSON#` since the percentage of working time depends on the project either. A relation is well-designed, if for each collection of attributes we have: Either all (other) attributes are functionally dependent on that collection or no (other) attribute is functionally dependent on that collection. This requirement, called *Boyce/Codd-normal-form* (*BCNF*) [Dat90], is violated by the relation *project administration* (as indicated above).
Though we did not give a fully formal definition of BCNF, it should be clear that it is only based on the concept of *functional dependency* which has a clear intuition and can be easily explained.

3 Workflow theory

In this section, we will sketch a workflow theory introduced by van der Aalst [Aal97a, Aal97b, Aal98]. Its simplicity and its correspondence to intuitive ideas are analogous to Codd's database theory. Moreover, this workflow theory helps to identify good workflows. Note, that this workflow theory is only concerned with the *behavioural* or *process* aspect of workflows — of course theories for the other aspects are necessary. Since the behavioural aspect is an important aspect of workflows, it is justified to call it workflow theory.
Let us consider an example first. A workflow (resp. the behavioural aspect) can be modelled by a Petri net (a Place/Transition-System, to be precise) with a distinguished *start* place and a distinguished *end* place. Moreover, each place except *start* is initially unmarked. A transition of the Petri net corresponds to an *activity* of the workflow; a place (or rather a token on a place)

corresponds to a *document*. The arcs of the Petri net indicate which documents are consumed and which are produced by an activity.

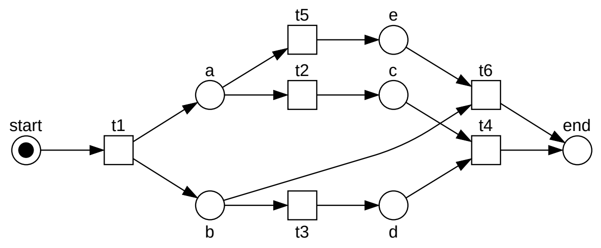

Figure 2: Workflow net 1

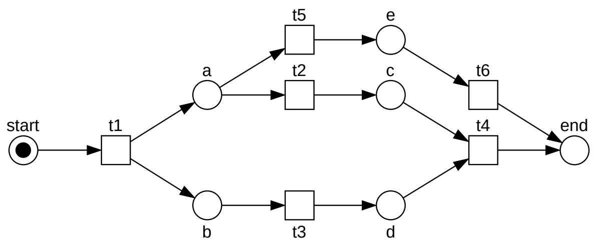

Figure 3: Workflow net 2

Figure 2 shows an example of an ill-designed workflow. We will point out some anomalies of this workflow. Suppose, activities $t1$, $t3$, and $t5$ have been processed already. Then, we have documents (tokens on places) e and d; in this situation no activity can be executed any more. Therefore, it is impossible to finish the workflow correctly by putting a document (token) on place *end*. Next, we try to fix this anomaly by removing the arc from b to $t6$ as shown in Fig. 3. Then, activity $t6$ can be processed in the above situation and the workflow terminates. This workflow, however, shows another anomaly: When the workflow terminates, there is still a document d within the workflow. This corresponds to the loss of a document within the workflow, which is not satisfactory. Another anomaly would be if there was an activity which is dead — i.e. it is impossible to ever perform this activity. These three anomalies are excluded in so-called *sound workflows* [Aal97a, Aal97b].
Up to now, we established an analogy to the anomalies of database theory. Next, we will give an analogy to the concept of functional dependency. Here, two classical concepts of Petri net theory come in: *liveness* and *boundedness* (cf. [Pet81, Rei85]). Informally, a Petri net is *live* if every transition can occur again whatever happened before; a Petri net is *bounded* if the number of tokens on each place is bounded by some number.
Now, these concepts can be used to characterize sound workflows: A workflow is sound, if and only if the Petri net extended by one transition between *end* and *start* as indicated in Fig. 4 is live and bounded [Aal97a, Aal97b].

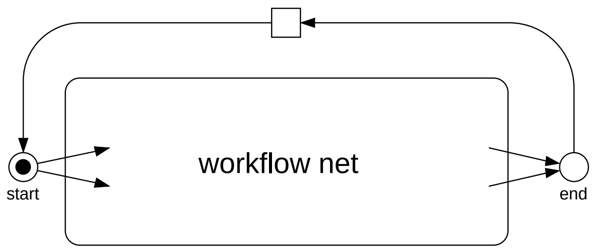

Figure 4: Extended workflow net

Note that in a sound workflow it is still possible that a document occurs twice on the same place (technically speaking, boundedness does not imply 1-safeness). From a practical point of view this is debatable. Therefore, we introduce a slightly stronger concept of soundness: A workflow net is *strong sound*, if and only if the extended workflow net is *live* and *1-safe*. In addition to the practical motivation that we do not want duplicate documents, we will give a theoretical justification for this stronger requirement at the end of the next section.

In analogy to Codd's theory, the above workflow theory helps to identify ill-designed workflows. The soundness criterion is motivated by anomalies in a similar way Codd motivates his normal forms. Liveness and boundedness are almost as simple as Codd's functional dependencies.

4 Workflow construct theory

The workflow theory as presented in the previous section helps to identify ill-designed workflows. But, it is only possible to check for soundness at the end of the design process; there are no guidelines for designing sound workflows. Here, we will briefly present another part of workflow theory which tackles this problem.

One way to enforce the design of good workflows is to construct workflows by so-called routing constructs. The *Workflow Management Coalition* (WfMC) proposes four routing constructs: *sequential*, *alternative*, *parallel*, and *iterated* execution. These routing constructs can be rephrased in terms of the transformation rules shown in Fig. 5. Van der Aalst [Aal97b] has shown that each

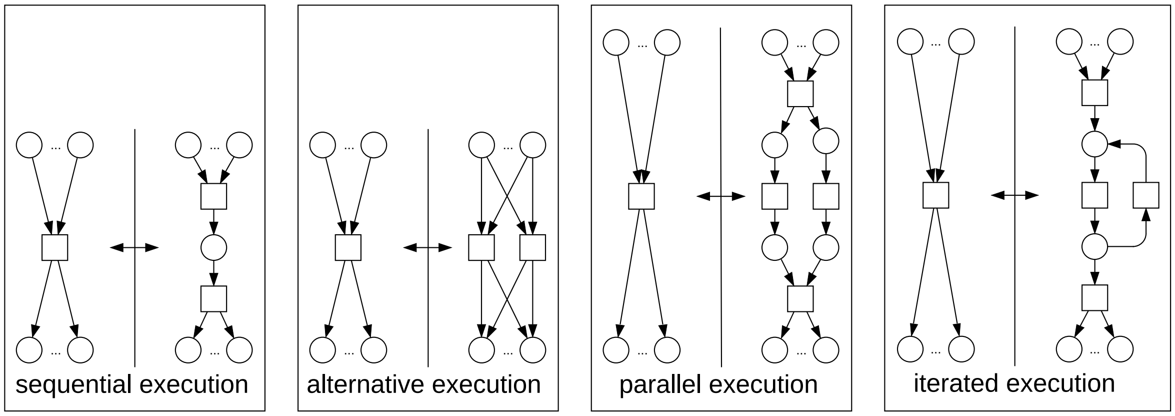

Figure 5: Routing constructs and transformation rules

transformation[1] preserves soundness of a workflow. In particular, any workflow net constructed by the above transformation rules from the workflow net shown in Fig. 6 is sound; indeed, it is strongly sound.

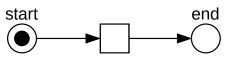

Figure 6: Workflow for starting the construction

The problem, however, is that the routing constructs of the WfMC are rather restrictive. Therefore, we need a theory to design additional routing constructs. Most interesting, this theory has already been proposed by Valette [Val79] — at a time when workflow was not an issue! When we represent a routing construct by a transformation rule as shown above, his result basically says: We can use any strongly sound workflow net as a workflow construct — i.e. as right-hand side of the transformation rule.

The above condition even guarantees that the workflows built from the new routing constructs are strongly sound. What is more, it is even a necessary and sufficient condition for routing constructs which guarantee *strong sound* workflows. This exact correspondence is a theoretical justification for strong soundness.

[1]Note that we have slightly changed the rule for iteration.

5 Conclusion

The technical contribution of this paper is rather modest: We have proposed a slightly modified soundness criterion for workflow nets.

The main purpose of this paper is to give a flavour of a future workflow theory based on the ideas of van der Aalst. Though it might appear trivial from a theoreticians point of view, it is just the bit of theory needed. As a justification we have compared his approach to Codd's doubtless successful database theory.
We do not claim that the development of this workflow theory is finished; but we claim that future workflow theory will consist of many results similar to those mentioned in this paper. Slight variations, e.g. strong soundness instead of soundness, may turn out to be more suitable; even this variation has a counterpart in Codd's theory: 3NF as defined by Codd has later been replaced by BCNF [Dat90].
What is more, we have given a new interpretation to Valette's [Val79] results: A criterion for defining new workflow constructs. We believe that more old Petri net results have a workflow interpretation and need just to be rediscovered and given a new interpretation. There is only little need for new Petri net theory, but a great need for Petri net archaeology.

Acknowledgments I would like to thank Wil van der Aalst for his tutorials on workflow, a day of discussion on his approach, and his comments on a preliminary version of this paper. I would also like to thank those theoreticians who argued that Wil's approach is no theory — their arguments inspired me to think of a striking analogy and to write this paper. Moreover, many thanks to Wolfgang Reisig and to my colleagues Rolf Walter, Sibylle Peuker, Tobias Vesper, Hagen Völzer, and Michael Weber who gave helpful hints for improving the presentation of this paper.

References

[Aal97a] W.M.P. van der Aalst. Exploring the process dimension of workflow management. Computing Science Reports 97/13, Eindhoven University of Technology, September 1997.

[Aal97b] W.M.P. van der Aalst. Verification of workflow nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets*, *LNCS* 1248, pages 407–426. Springer-Verlag, June 1997.

[Aal98] W.M.P. van der Aalst. Finding errors in the design of a workflow process: A Petri net approach. Unpublished manuscript, received January 1998.

[Cod90] E.F. Codd. *The Relational Model for Database Management, Version 2.* Addison-Wesley, 1990.

[Dat90] C.J. Date. *An Introduction to Database Systems, Volume I.* The Systems Programming Series. Addison-Wesley, 1990.

[Jab95] Stefan Jablonski. *Workflow-Management-Systeme: Modellierung und Architektur.* Thomson Publishers, 1995.

[Pet81] James L. Peterson. *Petri Net Theory And The Modeling of Systems.* Prentice-Hall, 1981.

[Rei85] Wolfgang Reisig. *Petri Nets*, *EATCS Monographs on Theoretical Computer Science* 4. Springer-Verlag, 1985.

[Val79] Robert Valette. Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, 18:35–46, 1979.