Pi calculus versus Petri nets: Let us eat "humble pie" rather than further inflate the "Pi hype"

W.M.P. van der Aalst

Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands. w.m.p.v.d.aalst@tm.tue.nl

Abstract. In the context of *Web Service Composition Languages* (WS-CLs) there is on ongoing debate on the best foundation for *Process-Aware Information Systems* (PAISs): *Petri nets* or *Pi calculus*. Example of PAISs are Workflow Management (WFM), Business Process Management (BPM), Business-to-Business (B2B), Customer Relationship Management (CRM), Enterprise Resource Planning (ERP) systems. Clearly, the web-service paradigm will change the architecture of these systems dramatically. Therefore, triggered by industry standards such as SOAP, WSDL, UDDI, etc., standards are being proposed for orchestrating web services. Examples of such WSCLs are BPEL4WS, BPML, WSFL, WSCI, and XLANG. In the debate on Petri nets versus Pi calculus many players in the "WSCL marketplace" are using demagogic arguments not based on concrete facts. This short note is an attempt to get to a more mature discussion on the pro's and con's of Petri nets and Pi calculus for WSCLs. A simple example is given to illustrate fundamental differences between Petri nets and Pi calculus. The paper also states *seven challenges*, in particular for those advocating the use of Pi calculus. Hopefully, this note will contribute to exposing the people that try to "hype" things like Pi calculus only for marketing purposes. Note that the big discrepancy between the "Pi-hype" and reality will not only limit the applicability of WSCLs but also discredit a beautiful scientific framework like Pi calculus.

Key words: Web Service Composition Languages, Petri nets, Pi calculus, BPEL4WS, BPML, WSFL, XLANG, and WSCI.

1 Introduction

The recently released BPEL4WS (Business Process Execution Language for Web Services, [14]) specification builds on IBM's WSFL (Web Services Flow Language, [24]) and Microsoft's XLANG (Web Services for Business Process Design, [30]). XLANG is a block-structured language with basic control flow structures such as sequence, switch (for conditional routing), while (for looping), all (for parallel routing), and pick (for race conditions based on timing or external triggers). In contrast to XLANG, WSFL is not limited to block structures and allows for directed graphs. The graphs can be nested but need to be acyclic. Iteration is only supported through exit conditions, i.e., an activity/subprocess is iterated until its exit condition is met. The control flow part of WSFL is almost identical to the workflow language used by IBM's MQ Series Workflow.

BPML (Business Process Modeling language, [11]) is a standard developed and promoted by BPMI.org (Business Process Management Initiative). BPMI.org is supported by several organizations, including Intalio, SAP, Sun, and Versata. The Web Service Choreography Interface (WSCI, [10]) submitted in June 2002 to the W3C by BEA Systems, BPMI.org, Commerce One, Fujitsu Limited, Intalio, IONA, Oracle Corporation, SAP AG, SeeBeyond Technology Corporation, and Sun Microsystems. There is a substantial overlap between BPML and WSCI.

While languages like BPEL4WS and BPML have been developed with web services in mind, the traditional workflow languages and tools made similar attempts not necessarily based on standards like SOAP, WSDL, UDDI, etc. An example is XML Process Definition Language (XPDL), the language proposed by the Workflow Management Coalition (WfMC) to interchange process definitions between different workflow products. The goal of XPDL is to provide a Lingua Franca for the workflow domain allowing for the import and export process definitions between a variety of tools ranging from workflow management systems to modeling and simulation tools.

The competition between these languages triggered a discussion on "the best" foundation for *Web Service Composition Languages* (WSCLs). Although few articulate the need for a formal basis clearly, the general opinion is that some formal model should be used to base these complex languages on. Clearly, formal models like Petri nets and Pi calculus offer advantages when it comes to providing solid semantics and analysis methods. Unfortunately, only in few cases such a foundation is really used to actually provide solid semantics and analysis methods. At this point in time, it seems that formal models are used to *advertise* languages rather than *to improve their quality and applicability*. Few examples such as the Woflan project [32] and the YAWL project [6] demonstrate in real-life situations the added value of formal methods. This brings us to the first two challenges.

Challenge 1: Let the people that advocate BPEL4WS, BPML, WSFL, XLANG, XPDL, and WSCI show the precise relation between the language and some formal foundation. People that cannot do this but still claim strong relationships between their language and e.g. Pi-calculus only cause confusion.

Challenge 2: Let the people that advocate a particular formal model (e.g. Pi-calculus) in the context of languages like BPEL4WS, BPML, WSFL, XLANG, XPDL, and WSCI demonstrate the use of analysis methods and tools based on this formal model (in some real life setting).

As indicated in the abstract, this paper is about the debate on Petri nets versus Pi calculus in the context of WSCLs. Therefore, some reflection on the history of concurrency and PAISs is in order.

2 History of PAISs and concurrency

Let us first focus on the history of PAISs. An interesting starting point from a scientific perspective is the early work on office information systems. In the seventies, Skip Ellis [15], Anatol Holt [19], and Michael Zisman [35] already worked on so-called office information systems, which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. During the seventies and eighties there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and related research almost stopped for a decade. Hardly any advances were made in the eighties. In the nineties, there was a renewed interest in these systems. The number of workflow management systems developed in the past decade and the many papers on workflow technology illustrate the revival of process-aware office information systems. Today workflow management systems are readily available. However, their application is still limited to specific industries such as banking and insurance. As indicated by Skip Ellis in [16] it is important to learn from these ups and downs. The failures in the eighties can be explained by both technical and conceptual problems. In the eighties, networks were slow or not present at all, there were no suitable graphical interfaces, and proper development software was missing. However, there were also more conceptual problems: there was no unified way of modeling processes and the systems were too rigid to be used by people in the workplace. Most of the technical problems have been resolved by now. However, the more conceptual problems remain. Good standards for business process modeling are still missing and even today's workflow management systems enforce unnecessary constraints on the process logic (e.g., processes are made more sequential that they need to be).

In [27] an interesting historic overview of office automation and workflow prototypes is given. History clearly shows that (i) workflow management is not something that started in the nineties but already in the seventies with the work of Ellis (*OfficeTalk*) and Zisman (*Scoop*); and (ii) the number of commercial systems has considerably grown in recent years. When considering WSCLs it is important to take this into account and use experience and knowledge from the workflow domain, i.e., do not re-invent the wheel!

Research in concurrency theory started with the PhD thesis of Carl Adam Petri [28]. Petri was the first person to develop models of interacting sequential processes. Before, his PhD thesis researchers only considered sequential processes represented in terms of transition systems or automata. About a decade later Robin Milner started working on concurrency theory. Later he would win the Turing award for the following three achievements: (1) LCF, the mechanization of Scott's Logic of Computable Functions, probably the first theoretically based yet practical tool for machine assisted proof construction; (2) ML, the first language to include polymorphic type inference together with a type-safe exception-handling mechanism; (3) CCS, a general theory of concurrency. The third achievement, i.e., the development of CCS, has been the main ingredient for

Pi calculus. In 1980 Robin Milner published the book "A Calculus of Communicating Systems" [25] describing the CCS language. CCS is one member of a large family of so-called *process algebra's*. Other established process algebra's include CSP [18] and ACP [12]. Pi calculus [26] is the most recent addition to the impressive collection of process algebra variants. It extends CCS with notions of mobility. While new process algebra's emerged Petri nets were extended with data (color), time, and hierarchy [20, 21]. See http://www.daimi.au.dk/PetriNets/ for more information on the various Petri net models.

The main difference between Petri nets and process algebra is that Petri nets are based on (bipartite) graphs while process algebra's are based on a textual (i.e., rather linear) description. In both areas there is an impressive accumulation of knowledge. Many notions developed for Petri nets have been translated to process algebra and vice versa. However, fundamental differences remain. For example, the notion of invariants developed for Petri nets [29] does not exist in process algebra. See [13] for a detailed comparison of Petri nets and process algebra.

Challenge 3: Let the people that advocate a particular formal model in the context of WSCLs actually study literature before making any statements.

One of the big misconceptions about Petri nets versus process algebra's is that process algebra's are compositional while Petri nets are not. *This is complete nonsense!* Petri nets have been extended with hierarchy. Moreover, Petri nets can be used in a compositional way. However, Petri nets also allow you to model in a non-compositional way. For example, by restriction subprocesses to e.g. WF-nets [1] similar compositionality results can be obtained [2, 9, 17].

3 An example

To start a more mature discussion on Petri nets versus process algebra, I propose to use concrete examples. To start such a discussion, I would like to use the model shown in Figure 1. This model shows a simple classical Petri net with 8 transitions. First a is executed followed by b and e in parallel. b is followed by c, however, f has to wait for the completion of both e and c, etc. Finally, h is executed and all transitions have been executed exactly once. Although the Petri net is very simple, e.g., it does not model any choices, only parallelism. Nevertheless, process algebra's like Pi calculus have problems modeling this simple example.

To understand the problem consider the Petri net shown in Figure 1 without the connection between c and f. In that case the sequences $b.c.d$ and $e.f.g$ are executed in parallel in-between a and h. In terms of Pi calculus this is denoted as $a.(b.c.d|e.f.g).h$. In this notation the "." is used to denote sequence and the "|" denotes parallelism. Indeed this notation is elegant and allows for computer manipulation. Unfortunately, such a simple notation is not possible if the connection between c and f is restored. The linear language does not allow

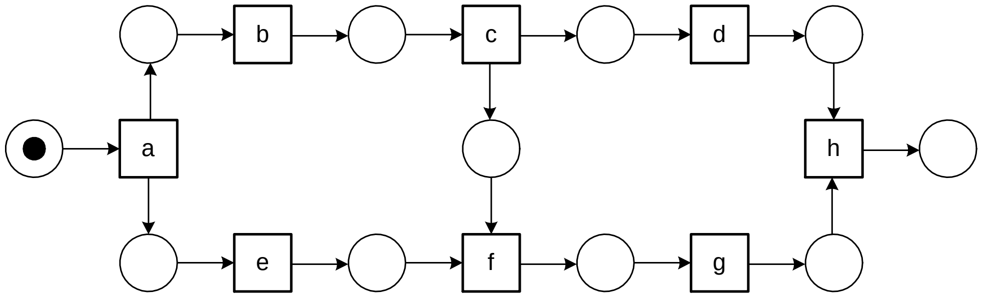

Fig. 1. How to model this in terms of Pi calculus?

for this while for a graph based language like Petri nets this is not a problem. Note that the claim is *not* that Pi calculus cannot model the process shown in Figure 1. However, it illustrates that Pi calculus is a language for experts where simple things suddenly become very complicated.

The example triggers two additional challenges.

Challenge 4: Let the people that advocate Pi calculus show how the Petri net shown in Figure 1 can be modeled easily.

Challenge 5: Let the people advocating Pi calculus propose modeling challenges for people advocating Petri nets as the fundamental language. It would be very interesting the see useful patterns that actually benefit from the notion of mobility present in Pi calculus.

Based on the outcome for the 5th challenge, it would be nice to try and map patterns involving mobility onto Petri nets. If needed, it would be interesting the apply the "nets in nets" paradigm developed by Valk et al. [31, 8] supported by tools like Renew (http://www.renew.de/).

4 Towards a more mature discussion

Clearly, Figure 1 is only a toy example. Therefore, we propose to use a set of relevant patterns to compare languages. Since 1999 we have been working on collecting a comprehensive set of workflow patterns [7]. The results have been made available through http://www.workflowpatterns.com, i.e., the "Workflow patterns WWW site". The patterns range from very simple patterns such as sequential routing (Pattern 1) to complex patterns involving complex synchronizations such as the discriminator pattern (Pattern 9). In this paper, we restrict ourselves to the 20 most relevant patterns. These patterns can be classified into six categories:

1. *Basic control-flow patterns.* These are the basic constructs present in most workflow languages to model sequential, parallel and conditional routing.

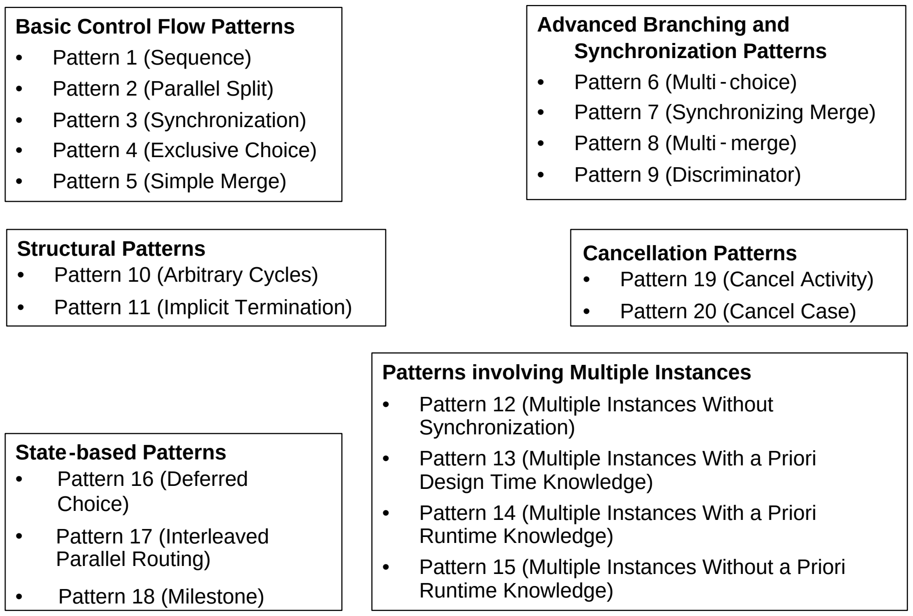
Basic Control Flow Patterns
• Pattern 1 (Sequence)
• Pattern 2 (Parallel Split)
• Pattern 3 (Synchronization)
• Pattern 4 (Exclusive Choice)
• Pattern 5 (Simple Merge)
Advanced Branching and Synchronization Patterns
• Pattern 6 (Multi-choice)
• Pattern 7 (Synchronizing Merge)
• Pattern 8 (Multi-merge)
• Pattern 9 (Discriminator)
Structural Patterns
• Pattern 10 (Arbitrary Cycles)
• Pattern 11 (Implicit Termination)
Cancellation Patterns
• Pattern 19 (Cancel Activity)
• Pattern 20 (Cancel Case)
Patterns involving Multiple Instances
• Pattern 12 (Multiple Instances Without Synchronization)
• Pattern 13 (Multiple Instances With a Priori Design Time Knowledge)
• Pattern 14 (Multiple Instances With a Priori Runtime Knowledge)
• Pattern 15 (Multiple Instances Without a Priori Runtime Knowledge)
State-based Patterns
• Pattern 16 (Deferred Choice)
• Pattern 17 (Interleaved Parallel Routing)
• Pattern 18 (Milestone)

Fig. 2. Overview of the 20 workflow patterns described in [7].

2. *Advanced branching and synchronization patterns.* These patterns transcend the basic patterns to allow for more advanced types of splitting and joining behavior. An example is the Synchronizing merge (Pattern 7) which behaves like an AND-join or XOR-join depending on the context.
3. *Structural patterns.* In programming languages a block structure which clearly identifies entry and exit points is quite natural. In graphical languages allowing for parallelism such a requirement is often considered to be too restrictive. Therefore, we have identified patterns that allow for a less rigid structure.
4. *Patterns involving multiple instances.* Within the context of a single case (i.e., workflow instance) sometimes parts of the process need to be instantiated multiple times, e.g., within the context of an insurance claim, multiple witness statements need to be processed.
5. *State-based patterns.* Typical workflow systems focus only on activities and events and not on states. This limits the expressiveness of the workflow language because it is not possible to have state dependent patterns such as the Milestone pattern (Pattern 18).
6. *Cancellation patterns.* The occurrence of an event (e.g., a customer canceling an order) may lead to the cancellation of activities. In some scenarios such events can even cause the withdrawal of the whole case.

Figure 2 shows an overview of the 20 patterns grouped into the six categories. A detailed discussion of these patterns is outside the scope of this paper. The interested reader is referred to [7] and `http://www.workflowpatterns.com`.

We have used these patterns to compare the functionality of numerous WFM systems but also most of the WSCLs. The result of this evaluation reveals that (1) the expressive power of contemporary systems/languages leaves much to be desired and (2) the systems support different patterns. Note that we do not use the term "expressiveness" in the traditional or formal sense. If one abstracts from capacity constraints, any workflow language is Turing complete. Therefore, it makes no sense to compare these languages using formal notions of expressiveness. Instead we use a more intuitive notion of expressiveness which takes the modeling effort into account. This more intuitive notion is often referred to as suitability. See [22, 23] for a discussion on the distinction between formal expressiveness and suitability.

We have evaluated the leading standards for WSCLs. See [33, 34] for more information about the evaluation of BPEL4WS, XLANG, and WSFL using the patterns. See [4] for more information about the evaluation of BPML and WSCI using the patterns. For an overview of these evaluations we refer to [3].

The observation that the expressive power of the available languages and systems leaves much to be desired, triggered the question: *How about Pi calculus?*

Challenge 6: Let the people that advocate Pi calculus exactly show how existing patterns can be modeled in terms of Pi calculus.

Challenge 7: Let the people advocating Pi calculus propose new patterns, especially patterns involving mobility.

5 Conclusion

This short note is an attempt to trigger a more mature discussion on the foundations of WSCLs. Both Petri nets and the Pi calculus are solid and respectable languages. Clearly, Robin Milner developed a beautiful language which can be applied in many application domains. However, the "Pi hype" is not based on any solid arguments. People that are not familiar with formal methods are fighting religious wars driven by commercial arguments instead of the desire to build a solid foundation for WSCLs. As a kind of "antidote" to the Pi hype, this paper proposed seven challenges. Moreover, to conclude I would like to discuss the pro's and con's of Petri nets as an alternative for Pi calculus.

There are at least three good reasons for using Petri nets as a basis for WSCL:[1]

1. *Formal semantics despite the graphical nature*
   On the one hand, Petri nets are a graphical language which allows for the modeling of the workflow primitives identified by the WfMC. On the other

[1] Note that we focus on the workflow-functionality of WSCL. Clearly, there are other aspects that are also important. However, the dominant perspective of WSCL is the workflow/process perspective. Only a superficial scan of existing WSCLs like BPEL4WS and BPML will reveal this.

hand, the semantics of Petri nets (including most of the extensions) have been defined formally. Many of today's available WFM systems provide adhoc constructs to model workflow procedures. Moreover, there are WFM systems that impose restrictions on many of the workflow patterns discussed. Some WFM systems also provide exotic constructs whose semantics are not 100% clear, cf. the join construct in XPDL and many other languages. Because of these problems it is better to use a well-established design language with formal semantics as a solid basis.

2. *State-based instead of event-based*
   In contrast to many other process modeling techniques, the state of case can be modeled explicitly in a Petri net. Process modeling techniques ranging from informal techniques such as dataflow diagrams to formal techniques such as process algebra's are *event-based*, i.e., transitions are modeled explicitly and the states between subsequent transitions are only modeled implicitly. Today's WFM systems are typically event-based, i.e., tasks are modeled explicitly and states between subsequent tasks are suppressed. The distinction between an event-based and a state-based description may appear to be subtle, but patterns like the Deferred choice (WP16) and the Milestone (WP18) show that this is of the utmost importance for workflow modeling.
3. *Abundance of analysis techniques*
   Petri nets are marked by the availability of many analysis techniques. Clearly, this is a great asset in favor of a Petri nets. Petri-net-based analysis techniques can be used to determine the correctness of a process designs. The availability of these techniques illustrates that Petri-net theory can be used to add powerful analysis capabilities to the next generation of PAISs.

However, as indicated in [5] there are also problems when modeling workflows in terms of a Petri nets. For the more advanced routing constructs it is necessary to resort to high-level nets [20, 21]. Moreover, a straightforward application of high-level Petri nets does not always yield the desired result. There seem to be three problems relevant for WSCLs:

1. High-level Petri nets support colored tokens, i.e., a token can have a value. Although it is possible to use this to identify multiple instances of a subprocess, there is no specific support for *patterns involving multiple instances* and the burden of keeping track, splitting, and joining of instances is carried by the designer.
2. Sometimes two flows need to be joined while it is not clear whether synchronization is needed, i.e., if both flows are active an AND-join is needed otherwise an XOR-join. Such *advanced synchronization patterns* are difficult to model in terms of a high-level Petri net because the firing rule only supports two types of joins: the AND-join (transition) or the XOR-join (place).
3. The firing of a transition is always local, i.e., enabling is only based on the tokens in the input places and firing is only affecting the input and output places. However, some events in the workflow may have an effect which is not local, e.g., because of an error tokens need to be removed from various places without knowing where the tokens reside. Everyone who has modeled such

a *cancellation pattern* (e.g., a global timeout mechanism) in terms of Petri nets knows that it is cumbersome to model a so-called "vacuum cleaner" removing tokens from selected parts of the net.

Compared to existing WFM languages high-level Petri nets are quite expressive when it comes to supporting the workflow patterns. Recall that we use the term "expressiveness" not in the formal sense. High-level Petri nets are Turing complete, and therefore, can do anything we can define in terms of an algorithm. However, this does not imply that the modeling effort is acceptable. High-level nets, in contrast to many workflow languages, have no problems dealing with state-based patterns. This is a direct consequence of the fact that Petri nets use places to represent states explicitly. Although high-level Petri nets outperform most of the existing languages when it comes to modeling the control flow, the result is not completely satisfactory since the three problems indicated hamper the application in the WFM/BPM domain. This triggered the development of *YAWL (Yet Another Workflow Language)*. YAWL is based on Petri nets but extended with additional features to facilitate the modeling of complex workflows [5, 6]. See `http://www.citi.qut.edu.au/yawl/` for more information or to download the YAWL system.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
3. W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.
4. W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. Pattern-Based Analysis of BPML (and WSCI). QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.
5. W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. In K. Jensen, editor, *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, volume 560 of *DAIMI*, pages 1–20, Aarhus, Denmark, August 2002. University of Aarhus.
6. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. Accepted for publication in *Information Systems*, and also available as QUT Technical report, FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2003.
7. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
8. W.M.P. van der Aalst, D. Moldt, and F. Wienberg. Enacting Interorganizational Workflows using Nets in Nets. In J. Becker, M zur Mühlen, and M. Rosemann, editors, *Proceedings of the 1999 Workflow Management Conference*, volume 70 of

Working Paper Series of the Department of Information systems, pages 117–136, Muenster, Germany, November 1999. University of Muenster.

9. W.M.P. van der Aalst, K.M. van Hee, and R.A. van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. *Science of Computer Programming*, 42(2-3):129–171, 2002.
10. A. Arkin, S. Askary, S. Fordin, and W. Jekel et al. Web Service Choreography Interface (WSCI) 1.0. Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems, 2002.
11. A. Arkin et al. Business Process Modeling Language (BPML), Version 1.0, 2002.
12. J.C.M. Baeten. *Procesalgebra : een formalisme voor parallelle, communicerende processen*. Kluwer, Deventer, 1986.
13. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998.
14. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2002.
15. C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.
16. C.A. Ellis and G. Nutt. Workflow: The Process Spectrum. In A. Sheth, editor, *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, pages 140–145, Athens, Georgia, May 1996.
17. K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.
18. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, 1985.
19. A. W. Holt. Coordination Technology and Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 278–296. Springer-Verlag, Berlin, 1985.
20. K. Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 342–416. Springer-Verlag, Berlin, 1990.
21. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
22. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows.* PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. Available via http://www.workflowpatterns.com.
23. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39(3):143–209, 2003.
24. F. Leymann. Web Services Flow Language, Version 1.0, 2001.
25. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.
26. R. Milner. *Communicating and Mobile Systems: The Pi-Calculus.* Cambridge University Press, Cambridge, UK, 1999.
27. M. Zur Muehlen. *Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems.* Logos, Berlin, 2004.

28. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Fakultät für Mathematik und Physik, Technische Hochschule Darmstadt, Darmstadt, Germany, 1962.
29. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
30. S. Thatte. XLANG Web Services for Business Process Design, 2001.
31. R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25. Springer-Verlag, Berlin, 1998.
32. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
33. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-Based Analysis of BPEL4WS. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.
34. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.
35. M.D. Zisman. *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.