A Comparison of (e)EPCs and UML 2 Activity Diagrams

Harald Störrle
Universität Innsbruck, Institut für Informatik
Technikerstrasse 21a, A-6020 Innsbruck, Österreich
Harald.Stoerrle@uibk.ac.at

Abstract: In this paper, Event Process Chains (EPCs) and activity diagrams (ADs) of the Unified Modeling Language (UML) are compared with respect to (1) their syntax and its expressiveness, (2) their semantic domains and problems, and (3) their pragmatics and application conditions. The comparison is based on industrial experience and a survey of the research literature. Our conclusion is that while earlier versions of the UML did not provide sufficient means for modeling of business processes, the current version does. Since UML provides additional benefits over EPCs when it comes to software development as a consequence of business process modeling, we predict that for these applications, UML ADs will prevail over EPCs.

1 Introduction

1.1 Motivation

In the Unified Modeling Language version 1.x (UML 1.x, [OMG03]), Activity diagrams (ADs) have played a very limited role, since they were only an alternative notation for the same concepts already defined for the state machine notation. With the recent version 2.0 of UML (UML 2, [OMG05, Stö05]) this has changed. The concept (i. e., meta class) of Activity has been introduced as a substrate for the semantics. The expressiveness of ADs has been strongly increased, and their semantics has been upgraded from the run-to-completion interleaving semantics of UML state machines to a "'Petri-like"' approach.

This change of the UML specification is motivated by the strong need for a notation within the UML framework that allows the modeling of business processes. This, however, is the main domain of Event Process Chains (EPCs, [Sch98, Sch95]), and one wonders how the two notations actually compare—can ADs take the place of EPCs in real world applications?

1.2 Approach

To answer this question, the syntax and its expressiveness, the semantic domains and problems, and the pragmatics and application conditions are examined for both notations. We

have not taken into account methodological aspects, in particular, the general approaches, that is, the methodology of ARIS and the associated tool support by the ARIS toolset are

1.3 Related Work

To our knowledge, there is no systematic in-depth comparison of EPCs and UML 2 ADs so far. Since UML 1.x is now obsolete, so are comparisons between EPCs and UML 1.x ADs. There are superficial comparisons like [Stö05, p. 252], and there might be internal concept papers in commercial organisations.

2 Syntax

EPCs are often considered an integral part of the ARIS method and toolset. The UML, on the other hand, is just a notation, not a method (as it was the case for UML predecessors like OMT and OOSE). By definition, methodological concerns are excluded from the subject domain belonging to UML. This separation of concerns is considered a major milestone in the UML community, since it allowed the standardized of the language independent of other, more complex issues. These need to be addressed in real life work too, of course, but focusing on one issue first has been helpful, as the tremendous success of UML over the last decade underlines.

In this paper, thus, we disregard methodological aspects. In this section, we start by comparing ADs and EPCs by their respective notational elements. This section is structured into basic syntax, data flow, exceptions, and complex nodes.

2.1 Basic Syntax

Both ADs and EPCs provide elementary actions, events, and parallel and optional control flows. Both notations allow procedure-call-like refinements. See Figure 1 and 2 for a tabular comparison of the basic syntax of ADs and EPCs.

Figures 1 and 2 makes it obvious, that ADs allow many more notational variants than EPCs. For instance, AD forks/joins have not only the basic logical operators and, or, xor, but arbitrary logical formulas. Similarly, arbitrary conditions may be imposed on case distinctions (decision nodes). ADs distinguish between the opening and closing of alternative and concurrent branches (though, unfortunately, the notation allows mixing the two), which facilitates the definition of well-nestedness of operators, a notorious problem for EPCs. In ADs, it is possible to distinguish between different kinds of final nodes (terminating the whole action vs. terminating only on concurrent flow), and different kinds of events (send, receive), while there is only the event-notion for start, stop, and all kinds of events in EPCs.

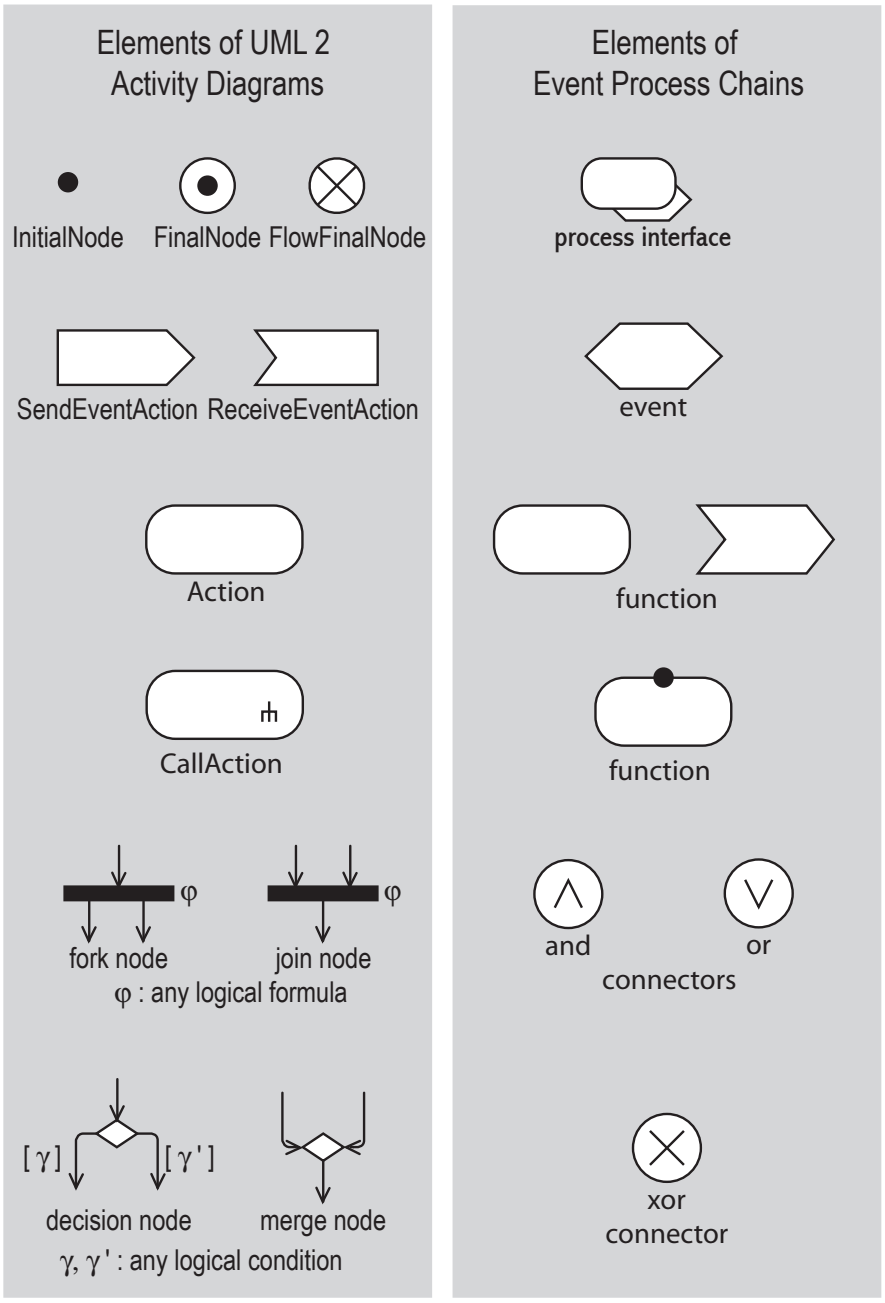

Figure 1: Comparing elementary ADs and EPCs: actions, events, and basic control nodes.

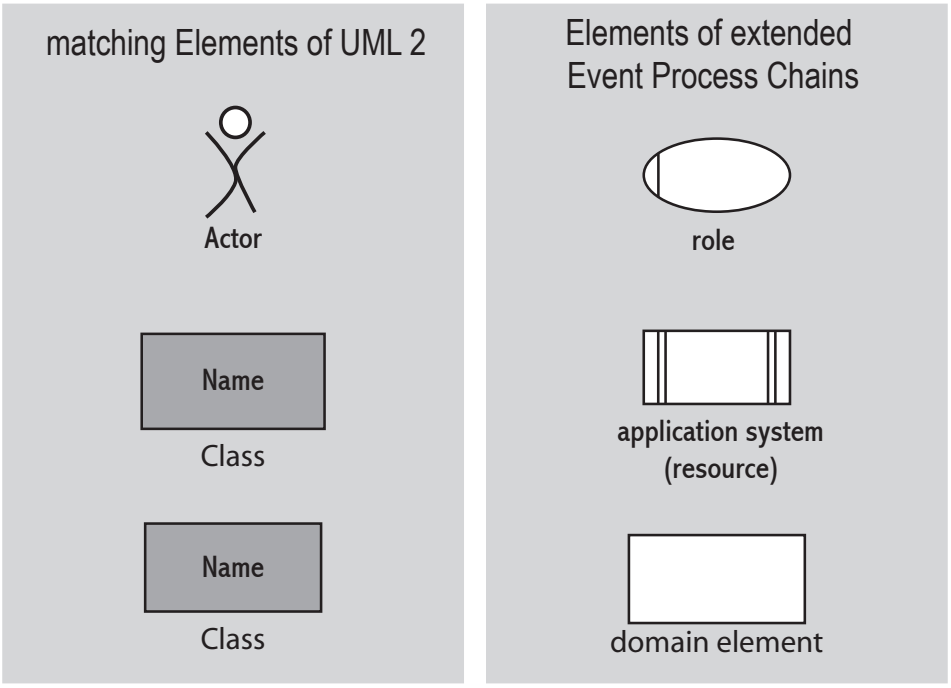

Figure 2: Comparing elementary ADs and eEPCs: actors, classes, and systems.

2.2 Data Flow

Regular EPCs do not provide facilities to model data flow, but extended EPCs (eEPCs) do, though only for the inputs and outputs of individual functions, not the flow as such. In ADs, on the other hand, a confusing variety of alternative notations with identical semantics is available (see Figure 3). Additionally, ADs provide means to model buffering strategies, weights, selection criteria and transformation actions for the edges connecting data and functions.

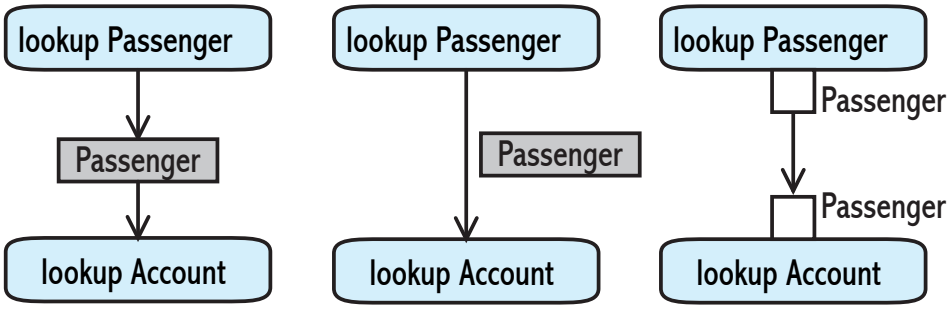

Figure 3: Notational variants fr data flow in UML 2 ADs: all three notations mean the same.

2.3 Exceptions

Probably the most novel feature of ADs are exceptions (see Figure 4), which are rather similar to the programming language concept. In practice, there are sometimes ill formed EPCs that misuse calls/refinements as a kind of GoTos (see Figure 5). This can and has been used to simulate exception-like behavior in an unstructured way. Of course, such models are very error-prone, and this kind of notational abuse should be prohibited.

2.4 Complex nodes

Another new concept of UML 2 ADs are so called structured activity nodes. They comprise structured nodes and expansion regions. Structured nodes correspond to some constructs of structured programming, most notably loops, and conditional. Expansion regions provide notations for different kinds of concurrent execution of actions (cf. [Stö04c]). There are no comparable concepts in the (e)EPC notation.

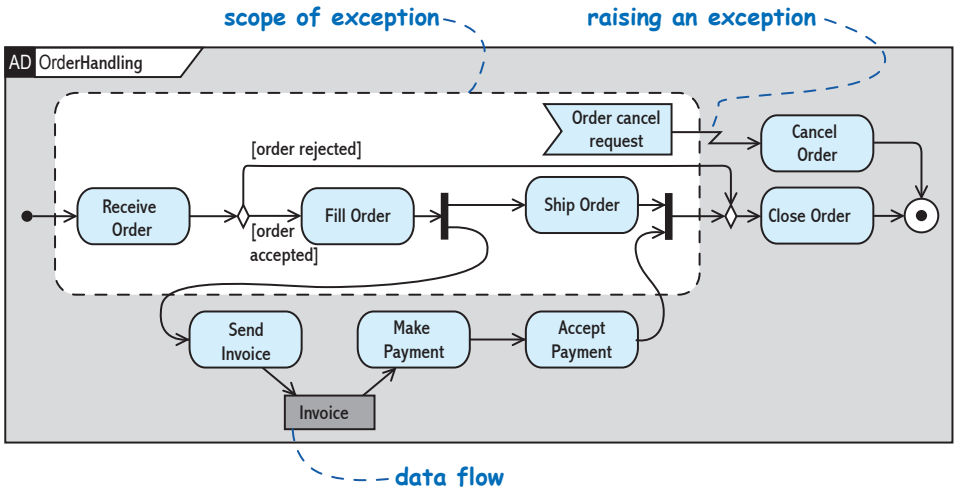

Figure 4: An AD with an exception and with data flow (taken from [OMG05]).

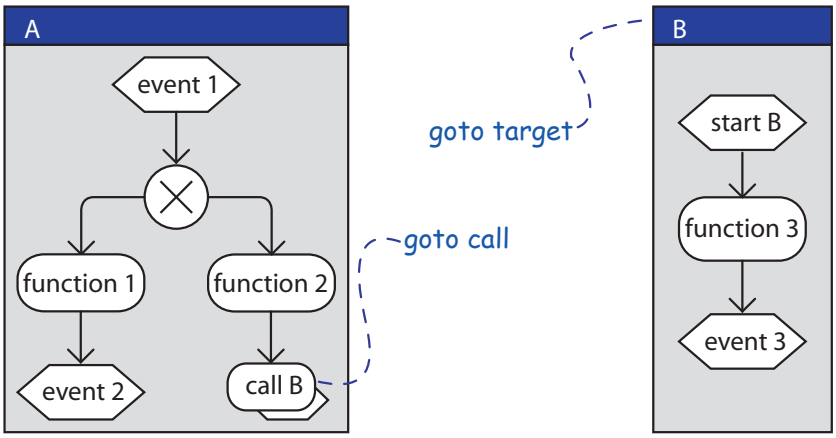

Figure 5: Abusing refinement/procedure call as goto.

2.5 Syntax comparison summary

All EPC concepts are available in ADs, often even in different variants. On the other hand, many AD concepts are not available in EPCs. The same is true for the annotations on elements. Therefore, the AD notation is much richer and thus much more expressive than the EPC notation.

3 Semantics

Both EPCs and ADs refer to the same semantic domain, namely Petri-nets. Both notations are defined in an informal way only, so that a formal semantics has to be defined separately. Many such approaches have been proposed for EPCs over the years (cf. e. g. [ADK03, DAV05]). For UML 1.x ADs, there have been some approaches, which are now obsolete, of course. For UML 2.0 ADs, not very much work has been done so far (in particular [Stö04b, Stö04a, BG04], or see e. g. the related work section in [Stö04c] for a complete summary).

Unfortunately, both notations imply a number of semantic problems, though this number is probably smaller for EPCs due to the smaller number of concepts and notational variants, which result in less trouble defining a formal semantics. Interestingly, some semantic problems occur for both notations in a very similar way (cf. the analogous findings in [CK04] and [SH05] on non-local semantics). It might be an interesting question to find out, whether a solution for one notation also works for the other.

In terms of formality, there is not much difference—both notations are informal. While the number of semantical issues seems to be smaller for EPCs, the syntax of ADs is more precisely defined due to the UML's meta modeling approach. Unless adequate formal semantics have been defined and generally agreed upon for both (e)EPCs and ADs, no comparison of the semantics is complete.

4 Pragmatics

4.1 Transition from analysis to implementation

In the syntax comparison above, we have concluded that ADs are much more expressive than EPCs. Concerning the pragmatics, notational variability is a mixed blessing, though: on the one hand, great expressiveness certainly is helpful to capture a wide range of processes more precisely. However, in the absence of a precise semantics, this might be a false precision. Also, one might suspect that the plenitude of concepts and notations in ADs make them more difficult to learn and comprehend. Whether this is an advantage for EPCs or for ADs depends on the application area in which they are used. There are a number of such potential application areas.

- analysis tasks like business process modeling
- design tasks like workflow modeling
- implementation tasks like program modeling

Business process modeling is the application domain EPCs have been created for (see [KNS92, LSW97]), and it has also been the most important driving force behind the enhancement and standardization of ADs in UML 2. Both notations have been used for this purpose in practice, though there are not very many experience reports on using UML 2 ADs yet. It is obvious though, that the notational richness is more of a hindrance for this type of task, due to the kind of people usually involved. However, it is of course possible to tailor ADs such that only the expressive means of EPCs are available. In fact, initial practical experience in industrial projects suggeststhat there are some notions and notations of UML 2 ADs that are quite natural for domain experts with little computer-science know how, such as the concept of exception. This seems to be very intuitive for many people.

Both EPCs and ADs have also been applied successfully to workflow modeling. Here, the notational richness of ADs is already an advantage, since there are design decisions that can not be represented in EPCs—one would have to enhance the notation or rely on a proprietary tool to support design tasks. Enhancements, of course, are much more difficult than restrictions.

The notational richness of ADs turns into a strong advantage when it comes to program modeling, that is, creating models on level of detail similar to programs. This could be done either constructively in order to create a running system or analytically in order to visualize programming language code such as BPEL, Java, and, of course, ABAP. Regular EPCs or extended EPCs are definitely too weak to be useful for such applications. It is of course possible to add more and more concepts and additional information to (e)EPCs, but then, the simplicity and ease of use that is their particular advantage would be lost, too.

Obviously, it is easier to go from analysis to design and to implementation using only one notation and only one toolset. This is the case for ADs which are a well integrated part of the UML. Thus it is very easy to proceed from a pure process description into other views concerned with data, domain architecture, software architecture and so on. Also, refining analysis level processes into design level processes is easier when the same notation is used for both.

Recent approaches to transform (e)EPCs into languages like the Business Process Execution Language (BPEL, [ACD+03]) or workflow definition languages, suffer from the same problem. While such approaches definitely improve the viability of using (e)EPCs as a front end to implementing service oriented architectures, they also imply another language interface with more opportunities for errors and more difficult traceability.

As a direct consequence of the previous considerations, it follows that ADs are better suited for software development projects than (e)EPCs. For business process reengineering purposes, on the other hand, using (e)EPCs has a small advantage in that no tailoring is required. This section is summarized in Figure 6.

NOTATION	APPLICABILITY IN LIFECYCLE PHASE		
	ANALYSIS	DESIGN	IMPLEMENTATION
(e)EPCs	$\surd$	($\surd$) (requires extension)	–
UML 2 ADs	($\surd$) (requires restriction)	$\surd$	$\surd$

Figure 6: Comparing EPCs and ADs by the phases of the software life cycle in which they are applicable.

4.2 Tools

Over the last 10 years, the number and quality of UML tools has increased dramatically. Today, there is a broad range of several hundred tools, from open-source to high-end industrial tool suites, from mere drawing tools to integrated development environments with team support, version and configuration control, code generation, consistency checkers, report generators and so on.

For (e)EPCs, on the other hand, there are only a few tools, most notably of course the ARIS toolset, but there is now also the EPC Tools open source initiative [CK]. Generally, EPC tools tended to also provide more advanced functionality like simulation and analysis of models which was absent in UML models.

With XMI, there has been a standardized data exchange format for many years. Even though standard compliance has been less than perfect, effective data exchange between different tools is a reality today for UML. The standardization of UML and its exchange format XMI are primarily driven by the Object Management Group (OMG). Since the OMG is just an industrial consortium its documents are not standards in the proper sense. However, the previous version of the UML (1.4.2) and the accompanying XMI version have been standardized by the International Standards Organisation (ISO) recently (ISO/IEC 19501:2005 and ISO/IEC 19503:2005), and the OMG also pursues the updating of these standards to more recent versions of UML.

For (e)EPCs on the other hand, there have been far less tools so that data exchange has not been as much of a problem as it had been for UML. Still, there has been demand for a general exchange format, and with EPML (cf. [MN04]), there is now a practical proposal, including tool support. This format seems to be widely accepted, though so far it lacks standardization even by the community or an industry consortium.

NOTATION	TOOL SUPPORT		
	NUMBER OF TOOLS	EXCHANGE FORMAT	STANDARDIZATION
(e)EPCs	~ 10	EPML	proprietary
UML 2 ADs	> 200	XMI	UML 1.4.2 / ISO 19501:2005 XMI / ISO 19503:2005

Figure 7: Comparing EPCs and ADs by the available tool support.

5 Discussion

For projects restricted to domain analysis, EPCs and tailored (i. e., simplified) ADs are about level-headed. Wherever code is to be produced at some point, however, ADs have an edge over EPCs as they allow seamless integration of analysis with design and implementation.

There are a number of general advantages ADs have over EPCs:

- UML ADs are part of a standard that is internationally accepted and developed, while EPCs are mainly used in Germany, and in particular in organisations using SAP software.
- There are so much more tools for ADs than there are for EPCs, that there is a great likeliness that there are better and cheaper UML tools than there are EPC tools. The more advanced functionalities traditionally associated with some EPC tools (e. g. simulation, model analysis, consistency checking and so on) are nowadays also found and often superseded in UML tools.
- Since UML is the de facto lingua franca of software engineering, all professional software engineers (and most computer scientists) will have had at least some exposure to UML, but frequently none to (e)EPCs. Therefore, UML ADs are a more viable choice in a software development project than (e)EPCs.
- Also, there is a much greater choice of books and commercial trainings, a larger body of scientific work, industrial experiences, and best practices for ADs than there is for EPCs.

None of these reasons is entirely compelling by itself—together, however, they make it likely that ADs will supersede (e)EPCs in the long run. Even for specialties of EPCs, where they currently still have advantages over ADs (certain tools, familiarity in certain communities etc.), ADs will spread to become the standard notation. How long this process may take remains to be seen.

References

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services (v1.1), 2003. available at `http://www.ebpml.org/bpel4ws.htm`.

[ADK03] Wil van der Aalst, Jörg Desel, and Ekkart Kindler. On the semantics of EPCs: A Framework for resolving the vicious circle (extended abstract). In Markus Nüttgens and Frank J. Rump, editors, *Proc. 2. Ws. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK'03)*, pages 71–79. Gesellschaft für Informatik e.V. 2003. available at `www.epk-community.de`.

[BG04] Conrad Bock and Michael Gruninger. PSL: A semantic domain for flow models. *Intl. J. Software and Systems Modeling*, Online First, 2004.

[CK] Nicolas Cuntz and Ekkart Kindler. The EPC Tools. available at `wwwcs.upb.de/cs/kindler/Forschung/EPCTools`.

[CK04] Nicolas Cuntz and Ekkart Kindler. On the semantics of EPCs: Efficient calculation and simulation. In Markus Nüttgens and Frank J. Rump, editors, *Proc. 3. Ws. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK'04)*, pages 7–26. Gesellschaft für Informatik e.V. 2004. available at `www.epk-community.de`.

[DAV05] B.F. van Dongen, Wil M.P. van der Aalst, and H.M.W. Verbeek. Verification of EPCs: Using Reduction Rules and Petri Nets. In Oscar Pastor and J. Falcao e Cunha, editors, *Proc. 17th Intl. Conf. Advanced Information Systems Engineering (CAiSE'05)*, number 3520 in LNCS, pages 372–386. Springer Verlag, 2005.

[KNS92] Gerd Keller, Markus Nüttgens, and August-Wilhelm Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerte Prozessketten (EPK)". Technical Report 89, Institut für Wirtschaftsinformatik, Uni Saarbrücken, 1992. available at `http://www.iwi.uni-sb.de/iwi-hefte/heft089.pdf`.

[LSW97] P. Langner, C. Schneider, and K. Wehler. Prozeßmodellierung mit ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen. *Wirtschaftsinformatik*, 39(5):479–489, 1997.

[MN04] Jan Mendling and Markus Nüttgens, editors. *Proc. 1st GI Ws. XML Interchange Formats for Business Process Management (XML4BPM)*, March 2004.

[OMG03] OMG. OMG Unified Modeling Language Specification (adopted formal specification, version 1.5). Technical report, Object Management Group, March 2003.

[OMG05] OMG. UML 2.0 Superstructure Specification (formal/05-07-04). Technical report, Object Management Group, August 2005. available at `www.omg.org`, downloaded at September 19^{th}, 2005.

[Sch95] August-Wilhelm Scheer. *Business Process Engineering. Reference Models for Industrial Enterprises*. Springer Verlag, 1995.

[Sch98] August-Wilhelm Scheer. *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. Springer Verlag, 3^{rd} edition, 1998.

[SH05] Harald Störrle and Jan Hendrik Hausmann. Obstacles on the Way Towards a Formal Semantics of UML 2.0 Activities. In Klaus Pohl, editor, *Proc. Natl. Germ. Conf. Software-Engineering 2005 (SE'05)*, number P-64 in Lecture Notes in Informatics, pages 117–128. Gesellschaft für Informatik e.V. 2005.

[Stö04a] Harald Störrle. Semantics and Verification of Data-Flow in UML 2.0 Activities. In Mark Minas, editor, *Proc. Intl. Ws. on Visual Languages and Formal Methods (VLFM'04)*, pages 38–52. IEEE Press, 2004. available at `www.pst.informatik.uni-muenchen.de/~stoerrle`.

[Stö04b] Harald Störrle. Semantics of Control-Flow in UML 2.0 Activities. In Paolo Bottoni, Chris Hundhausen, Stefano Levialdi, and Genny Tortora, editors, *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 235–242. IEEE Computer Society, 2004.

[Stö04c] Harald Störrle. Semantics of Expansion Nodes in UML 2.0 Activities. *Nordic Journal of Computing*, 11(3):1–24, 2004.

[Stö05] Harald Störrle. *UML 2 erfolgreich einsetzen*. Addison-Wesley, 2005.