A Generic Import Framework For Process Event Logs

Industrial Paper

Christian W. Günther and Wil M.P. van der Aalst

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

Abstract. The application of process mining techniques to real-life corporate environments has been of an ad-hoc nature so far, focused on proving the concept. One major reason for this rather slow adoption has been the complicated task of transforming real-life event log data to the MXML format used by advanced process mining tools, such as ProM. In this paper, the ProM Import Framework is presented, which has been designed to bridge this gap and to build a stable foundation for the extraction of event log data from any given PAIS implementation. Its flexible and extensible architecture, adherence to open standards, and open source availability make it a versatile contribution to the general BPI community.

1 Introduction

Process-Aware Information Systems (PAISs) are a commonplace part of the modern enterprise IT infrastructure, as dedicated process management systems or as workflow management components embedded in larger frameworks, such as Enterprise Resource Planning (ERP) systems.

At this point in time, most *business process monitoring* solutions focus on the performance aspects of process executions, providing statistical data and identifying problematic cases. The area of *Process Mining* [3], in contrast, is based on the *a-posteriori* analysis of process execution event logs. From this information, process mining techniques can derive abstract information about the different perspectives of a process, e.g. control flow, social network, etc.

There exists a great variety of PAIS implementations in field use, of which each one follows a custom manner of specifying, controlling and interpreting business processes. As an example, consider the utter difference in paradigm between a traditional, rigid Workflow Management System (WFMS) like Staffware on the one side, and a flexible case handling [5] system like FLOW*er* [7] on the other. This scale brings with it a corresponding plethora of event log formats, and concepts for their storage and accessibility.

In order to render the design of process mining techniques and tools independent of the target PAIS implementation, the *MXML* event log format has

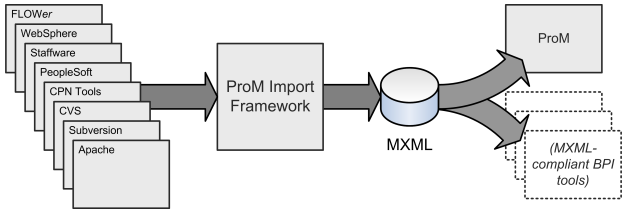

Fig. 1. Positioning the ProM Import Framework in the BPI landscape

been devised. While this format has been designed to meet the requirements of process mining tools in the best possible way, the conversion from many PAIS's custom formats to MXML is a non-trivial task at best.

This combination of recurring and time-consuming tasks calls for a generic software framework, which allows the implementation of import routines to concentrate on the core tasks which differentiate it from others. Providing a common base for a large number of import routines further enables to leverage the complete product with marginal additional implementation cost, e.g. by providing a common graphical user interface (GUI) within the host application.

The ProM Import Framework addresses these requirements, featuring a flexible and extensible plug-in architecture. Hosted import plug-ins are provided with a set of convenience functionality at no additional implementation cost, thus making the development of these plug-ins efficient and fast.

This paper is organized as follows. Section 2 introduces process mining and the ProM framework, followed by an introduction to the underlying MXML format in Section 3. Section 4 describes requirements, design, architecture, and implementation of the ProM Import Framework. Subsequently, Section 5 gives an overview about target systems for which import plug-ins have already been developed, after which Section 6 draws conclusions.

2 Process Mining and ProM

Process-aware information systems, such as WfMS, ERP, CRM and B2B systems, need to be configured based on process models specifying the order in which process steps are to be executed [1]. Creating such models is a complex and time-consuming task for which different approaches exist. The most traditional approach is to analyze and design the processes explicitly, making use of a business process modeling tool. However, this approach has often resulted in discrepancies between the actual business processes and the ones as perceived by designers [3]; therefore, very often, the initial design of a process model is incomplete, subjective, and at a too high level. Instead of starting with an explicit process design, process mining aims at extracting process knowledge from "process execution logs".

Process mining techniques such as the alpha algorithm [4] typically assume that it is possible to sequentially record events such that each event refers to

an activity (i.e., a well-defined step in the process) and to a case (i.e., a process instance). Moreover, there are other techniques explicitly using additional information, such as the performer and timestamp of the event, or data elements recorded with the event (e.g., the size of an order).

This information can be used to automatically construct process models, for which various approaches have been devised [6, 8, 11, 12]. For example, the alpha algorithm [4] can construct a Petri net model describing the behavior observed in the log. The Multi-Phase Mining approach [9] can be used to construct an Event-driven Process Chain (EPC) [14] based on similar information. At this point in time there are mature tools such as the ProM framework to construct different types of models based on real process executions [10].

So far, research on process mining has mainly focused on issues related to control flow mining. Different algorithms and advanced mining techniques have been developed and implemented in this context (e.g., making use of inductive learning techniques or genetic algorithms). Tackled problems include concurrency and loop backs in process executions, but also issues related to the handling of noise (e.g., exceptions). Furthermore, some initial work regarding the mining of other model perspectives (e.g., organizational aspects) and data-driven process support systems (e.g., case handling systems) has been conducted [2].

3 The MXML Format

The *MXML* format (as in *M*ining *XML*) is a generic XML-based format suitable for representing and storing event log data. While focusing on the core information necessary to perform process mining, the format reserves generic fields for extra information that is potentially provided by a PAIS.

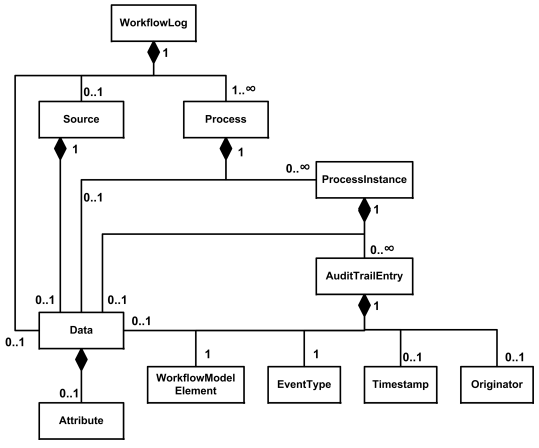

Fig. 2. Schema of the MXML format (UML diagram)

The structure of an MXML document is depicted in Figure 2, in the format of a UML 2.0 class diagram. The root node of each MXML document is a *WorkflowLog*, representing a log file. Every workflow log can potentially contain one *Source* element, which is used to describe the system the log has been imported from.

A workflow log can contain an arbitrary number of *Processes* as child elements. Each element of type "Process" groups events having occurred during the execution of a specific process definition. The single executions of that process definition are represented by child elements of type *ProcessInstance*. Thus, each process instance represents one specific case in the system.

Finally, process instances each group an arbitrary number of *AuditTrailEntry* child nodes, each describing one specific event in the log. Every audit trail entry must contain at least two child elements: The *WorkflowModelElement* describes the process definition element to which the event refers, e.g. the name of the task that was executed. The second mandatory element is the *EventType*, describing the nature of the event, e.g. whether a task was scheduled, completed, etc. Two further child elements of an audit trail entry are optional, namely the *Timestamp* and the *Originator*. The timestamp holds the exact date and time of when the event has occurred, while the originator identifies the resource, e.g. person, which has triggered the event in the system.

The elements described above provide the basic set of information used by current process mining techniques. To enable the flexible extension of this format with additional information extracted from a PAIS, all mentioned elements (except the child elements of *AuditTrailEntry*) can also have a generic *Data* child element. The data element groups an arbitrary number of *Attributes*, which are key-value pairs of strings.

4 The ProM Import Framework

While the ProM tool suite, which is based on interpreting event log data in the MXML format, has matured over the last couple of years, there is still a gap in *actually getting these logs* in the MXML format. Creating event logs in MXML has, in the past, been mostly achieved by artificial means, i.e. simulation, or by ad-hoc solutions which are not applicable to production use.

The *ProM Import Framework* steps in to bridge this gap. Its incentive is, on the one hand, to provide an adequate and convenient means for process mining researchers to actually acquire event logs from real production systems. On the other hand, it gives the owners of processes, i.e. management in organizations relying on PAIS operations, a means for productively applying process mining analysis techniques to their installations.

The following subsection introduces the incentives and high-level goals which have triggered the development of the ProM Import Framework. Section 4.2 derives from these goals a set of prominent design decisions, which form the basis of the system architecture, introduced in Section 4.3.

4.1 Goals and Requirements

In order to further progress the field of process mining it is essential to adapt and tailor both present and future techniques towards real-life usage scenarios, such that process mining can evolve into production use. This evolution fundamentally depends on the availability of real-life event log data, as only these can provide the necessary feedback for the development of process mining techniques.

Conversely, the process of actually applying process mining techniques in real world scenarios has to be eased and streamlined significantly. While several successful projects have proved the concept, it is a necessity to improve tool support for the entire process mining procedure from beginning to end.

A practical process mining endeavor is characterized by three, mainly independent, phases: At first, the event log data has to be imported from the source system. Secondly, the log data needs to be analyzed using an appropriate set of process mining techniques. Third and last, the results gained from process mining need thorough and domain-dependent interpretation, to figure out what the results mean in the given context, and what conclusions can be drawn.

The process mining specialist is required in the second and third phase, while the user, or process owner, is involved mainly in the third phase. What makes the first phase stick out is that it is at the moment the one task which can be performed with the least domain and process mining knowledge involved. Therefore, it is the logical next step for the progression of process mining to provide adequate and convenient tool support for the event log extraction phase.

A tool for supporting the event log extraction phase should thus address the following, high-level goals:

- The tool must be relatively *easy to operate*, such that also less qualified personnel can perform the task of event log extraction. This requirement implies, that:
  - By separating a configuration and adjustment phase from the extraction phase, which can potentially run unattended, the whole process can be leveraged and rendered more efficient.
- While ease of use is among the top goals, it must not supersede *flexibility and configurability* of the application. It must be applicable in as great an array of PAIS installations as possible.
- The tool must *provide an extensible and stable platform* for future development.
- It is advisable to *provide the tool on a free basis*, in order to encourage its widespread use and lower potential barriers for user acceptance. Further, *providing the code under an open source license* is expected to attract also external developers to participate. This enables the application to benefit from community feedback and contribution, thereby greatly leveraging the tool and, ultimately, process mining as a whole.

The subsequent subsection introduces the design decisions which were derived from these high-level goals.

4.2 Design Decisions

The ProM Import Framework has been developed from scratch, with the fundamental goal to provide a most friendly environment for developing import filters[1]. Consequently, a strong focus has been on extensibility and stability of the design, while including as much functionality as possible in the framework itself.

This emphasis has led to six crucial design choices, which have served as the cornerstone for developing the architecture of the system:

1. **Extensibility:** The design must incorporate a strict separation between general framework code and extension components. An additional requirement is to shift as much application logic as possible into the core framework, to prevent code duplication and to ease the development of extensions.
2. **Anonymization of log information:** The framework shall enable users to anonymize sensitive information contained in event logs in a transparent and convenient manner, thereby providing a means to protect the log owner's intellectual property.
3. **Flexible log-writing pipeline:** A logical *log-writing pipeline* shall be implemented, allowing to transparently chain a random number of log data-altering algorithms between event extraction and final storage.
4. **Decoupled configuration management:** It shall be sufficient for an import routine to specify its configuration options, and their corresponding types. Based on this information, the framework should transparently handle presenting these options to the user and allowing him to change them in a convenient manner.
5. **Decoupled dependencies management:** One further requirement towards the framework is to transparently satisfy all import routines' external requirements, e.g. database connectivity libraries.
6. **Convenient and decoupled user interface:** The application shall be relatively easy to use, i.e. it shall not require the user to have knowledge about process mining internals or the process of importing the event log data.

These design principles, together with the high-level goals presented in Section 4.1, have been used as an imperative in shaping the application's concrete architecture, which is presented in the following subsection.

4.3 Architecture

The architecture reflects the design principles stated in Section 4.2, and thus directly supports the high-level goals of Section 4.1. Figure 3 describes the abstract architecture of the framework, identifying the major classes involved and their mutual relationships in form of a UML 2.0 class diagram.

A flexible plug-in architecture satisfies the requirement for extensibility. For every target PAIS implementation, one dedicated import filter is supposed to

[1] The distribution is available at http://promimport.sourceforge.net.

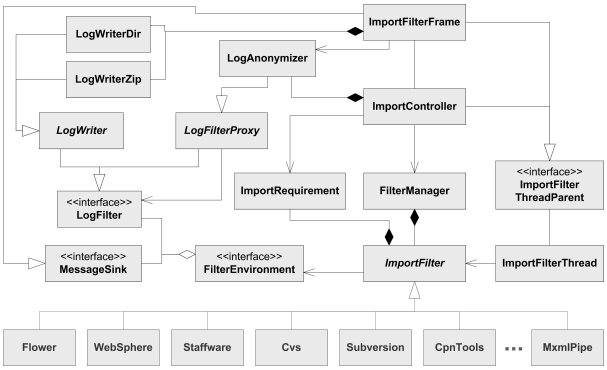

Fig. 3. Architecture of the ProM Import Framework, core components (UML diagram)

be implemented as a plug-in. Each import filter plug-in is contained within one dedicated class, derived from the abstract superclass *ImportFilter*. From this base class, every import filter plug-in inherits a set of methods which it can call in its constructor, to notify the system of its configuration options and external dependencies. For the actual import routine, the plug-in is passed an object implementing the interface *FilterEnvironment*, connecting the import filter to fundamental framework capabilities during the import procedure.

All elements of the log-writing pipeline implement the *LogFilter* interface, which allows for their flexible arrangement within the pipeline at will. This interface is used in a sequential manner, i.e. it incorporates methods to start and finish log files, processes and process instances, and a method for passing audit trail entries. The final endpoint of the log-writing pipeline is marked by an object derived from the abstract class *LogWriter* providing basic MXML formatting, while actual writing to permanent storage is implemented in LogWriter's subclasses.

Intermediate elements of the log-writing pipeline, such as the *LogAnonymizer*, are derived from the abstract class *LogFilterProxy*, implementing their transparent integration into the pipeline. At this point in time the anonymizer component is the only intermediate pipeline transformer available.

The *FilterManager* groups the set of import filters, provides named access to them, and provides their configuration within the framework for abstract access and modification. The *ImportController*, which incorporates the filter manager, manages the persistency of configuration data for the whole application and transparently manages and satisfies import filters' external requirements.

The class *ImportFilterFrame* implements the main graphical user interface of the application, including basic user interaction logic.

4.4 Disk-buffered Event Sorting

The log writing pipeline in the framework expects process instances to be transmitted one after another, while audit trail entries are supposed to be transmitted

in their natural order (i.e., order of occurrence). As not all import routines can expect their events in an ordered fashion, the framework provides the plug-in developer with a simple interface for transmitting unsorted event data, while ensuring that the sorting takes place in a transparent, resource-efficient manner.

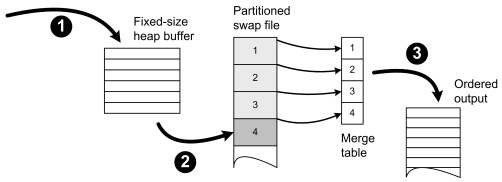

Fig. 4. Disk-buffered sorting in the framework

As this concept implies that all audit trail entries of an import session have to be buffered, before the first of them can be written, the process instance buffers are implemented to swap their content partially to disk storage.

This disk-buffered sorting mechanism is described in Figure 4.

1. Every buffer is equipped with a fixed-size buffer residing in heap space. This heap buffer is filled, as new audit trail entries are added to the process instance buffer.
2. When the heap buffer is completely filled with audit trail entries, it needs to be *flushed*. First, the events contained within the heap buffer are sorted using a Quicksort [13] algorithm. Then, all events in the heap buffer are appended to a swap file. Thus, the swap file contains subsequent segments, of which each contains a fixed number of sorted audit trail entries corresponding to one flush operation.
3. After all events have been received, the buffer needs to be emptied into the log writing pipeline in a sorted manner. An array called the *merge table*, with one cell per flush segment in the swap file, is initially filled with the first audit trail entry from each segment. Then, a modified merge sort [15] algorithm picks the first (in terms of logical order) event from the merge table, writes it to the log writing pipeline, and replaces it with the next entry from the respective flush segment in the swap file. This procedure is repeated, until all audit trail entries from the swap file have been loaded and the merge table is empty.

The presented disk-buffered sorting mechanism manages to effectively limit memory usage of the application. At the same time, a performance lag due to disk I/O is minimized by pre-buffering and sorting events in the heap buffer. Note that the algorithm scales well with the degree, in which incoming audit trail entries are already ordered. The less audit trail entries are in wrong order, the faster the initial sorting can be performed.

4.5 User Interface

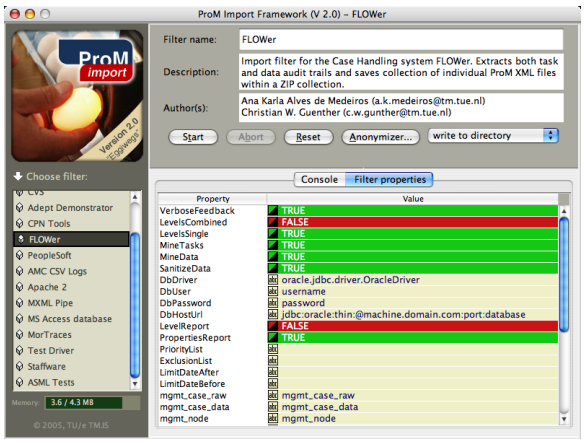

Fig. 5. User interface of the ProM Import Framework

The graphical user interface, which is depicted in Figure 5, is kept rather simple. On the left, an overview list allows the user to pick the import filter plug-in to be used. The upper right part shows general import filter properties, such as name, description, and author. Further, this part includes controls for the import procedure and the log anonymizer component.

The lower right part of the interface can either display a console view, or a configuration pane allowing to modify configuration settings for import filters. When the import procedure is started, the view switches to show the console, which is used to display feedback and error messages to the user.

5 Target Systems

The number of target systems, for which import plug-ins have been developed, has been steadily growing and diversifying since the development of the ProM Import Framework began[2]. On the one hand, this development is driven by advances in industry and practice, making ever more real-life PAIS implementations available for process mining research. On the other hand, this research triggers new applications from within, thus extending the field of "interesting" target systems.

In both directions, the flexible and extensible architecture of the ProM Import Framework has allowed developers to quickly implement solid and versatile solutions, taking advantage of the broad set of support functionality and clean

[2] The current distribution of the framework, including all plug-ins, can be downloaded from http://promimport.sourceforge.net.

user interface which the framework provides. At the time of this writing, there exist import plug-ins for the following target systems:

FLOW*er*: This product is an implementation of the *case handling* paradigm, which represents a very flexible, data-driven approach within the greater family of workflow management systems.

WebSphere Process Choreographer: As a part of IBM's WebSphere suite, the Process Choreographer is used to implement high-level business processes, based on the BPEL language.

Staffware: A workflow management system in the traditional sense, which has an impressive market coverage.

PeopleSoft Financials: Part of the PeopleSoft suite for Enterprise Resource Planning (ERP), this module is concerned with financial administration within an organization.

CPN Tools: CPN Tools provides excellent tool support for modelling Colored Petri Nets (CPN), a family of high-level Petri Nets, including a simulation engine for executing models. An extension to CPN tools has been developed, allowing to create synthetic event logs during a model simulation.

CVS: The process of distributed software development, as reflected in the commits to a source code repository like CVS, can also be analyzed with techniques from the process mining family.

Subversion: The Subversion system addresses fundamental flaws present in CVS, providing change logs that can also be interpreted by means of process mining.

Apache 2: As the access logs of web servers, like Apache 2, reveal the identity of users from their IP, the exact time and items requested, it is straightforward to distill process event logs from them.

As diverse as this list may read, it shows the impressive capabilities of the framework in enabling rapid development of import capabilities. The complexity of demanding import filters is significantly reduced by standard functionality offered by the framework. On top of that, the existence of a powerful framework allows for rapid prototyping of event log import capabilities.

Thereby it stimulates and supports experiments with less obvious systems, which may otherwise have been deemed not worth the effort. These can serve as effective and efficient means to evaluate the feasibility and usefulness of an import effort. An excerpt of ad-hoc solutions to import custom data sets, which were rapidly and successfully implemented using the ProM Import Framework, includes:

- Import of event logs describing the process of patient treatments from raw database tables provided by a Dutch hospital.
- Production unit test logs from an international manufacturer of IC chip production equipment.
- Conversion of spreadsheets containing patient treatment processes, from an ambulant care unit in Israel and a large Dutch hospital.
- Versatile and highly configurable import from the WFMS Adept [16], which is known for its rich set of features addressing flexibility.

6 Conclusions

The MXML format is the most widely adopted standard for the storage of process event logs in process mining research. This is most notably due to the fact that the ProM framework, providing a wide selection of process mining analysis techniques, relies on MXML for reading event logs.

However, due to a lack of convenient conversion tools, the availability of real-life event logs in MXML format has not been satisfactory so far. On the one hand, this lack of actual logs had a serious impact on the credibility of process mining techniques with respect to real-life applications. On the other hand, these techniques could not be used to analyze and improve industrial processes, and could thus not be put to use in real-life organizations.

In this paper, we have presented the ProM Import Framework, which is effectively bridging this gap. It represents a typical *enabling technology*, connecting formerly separate areas to their mutual benefit. In its current release, this application already features import plug-ins supporting seven process-aware information systems. Most notably, the support for commercial systems like FLOW*er*, WebSphere, and Staffware covers an immense installed base of users. Additional functionality that has been shifted into the framework makes the development of additional import plug-ins a convenient, time-effective task.

We hold this extension to the process mining tool landscape to be crucial with respect to the quality and credibility of process mining research. Real-life event log data often exhibits awkward and strange properties, which are unforeseen on a theoretical level, and which have to be taken into account in order to obtain meaningful results. It is only after process mining techniques have been proven to successfully analyze real-life logs, and thus to benefit businesses in their daily operations, that these techniques can grow into productive tools for business process optimization.

7 Acknowledgements

This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

References

1. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2002.
2. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer-Verlag, Berlin, 2004.

3. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
4. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
5. W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
6. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
7. Pallas Athena. *Case Handling with FLOWer: Beyond workflow.* Pallas Athena BV, Apeldoorn, The Netherlands, 2002.
8. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
9. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.
10. B.F. van Dongen, A.K. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The prom framework: A new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
11. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
12. J. Herbst and D. Karagiannis. An Inductive Approach to the Acquisition and Adaptation of Workflow Models. In M. Ibrahim and B. Drabble, editors, *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52–57, Stockholm, Sweden, August 1999.
13. C.A.R. Hoare. Algorithm 64: Quicksort. *Commun. ACM*, 4(7):321, 1961.
14. G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.
15. D.E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison Wesley, Reading, MA, USA, 2 edition, 1998.
16. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.