A Generic Import Framework For Process Event Logs

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 so far been of an ad-hoc nature, 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 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 Staffwareon 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 process mining techniques and tools independent of the target PAIS implementation, the *MXML* event log format has been devised. While this format has been designed to meet the requirements of process

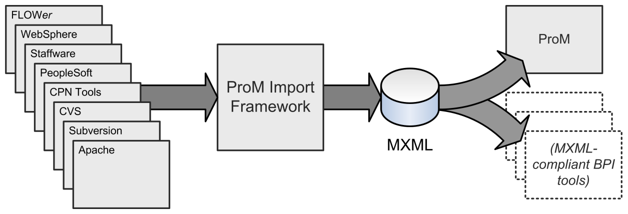


Fig. 1. Positioning the ProM Import Framework in the BPI landscape

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, 12, 13]. For example, the alpha algorithm [4] can construct a Petri net model describing the behavior observed in the log. The Multi-Phase Mining approach [10] can be used to construct an Event-driven Process Chain (EPC) [15] 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 [11].

So far, research on process mining has mainly focussed 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.

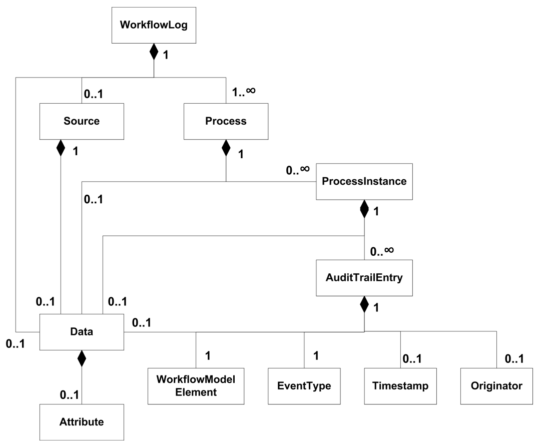


Fig. 2. Schema of the MXML format (UML diagram)

The structure of an MXML document is depicted in Figure 2, in the format of an 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 been 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* elements as 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, program or component, 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 string-based pairs of one key and an associated value each, thus providing a flexible and extensible means to capture name-referenced, additional information.

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 subsequent 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 section 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 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 subsequent section.

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.

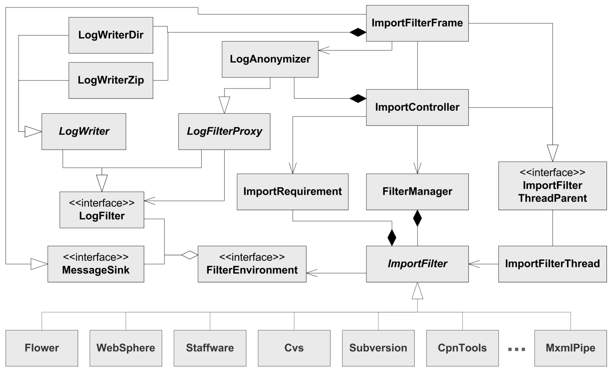


Fig. 3. Architecture of the ProM Import Framework, core components (UML diagram)

be implemented in form of 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.

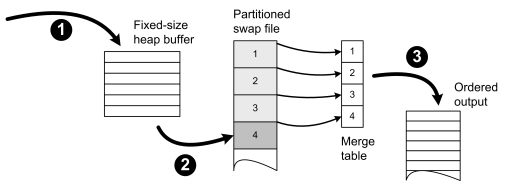


Fig. 4. Disk-buffered sorting in the framework

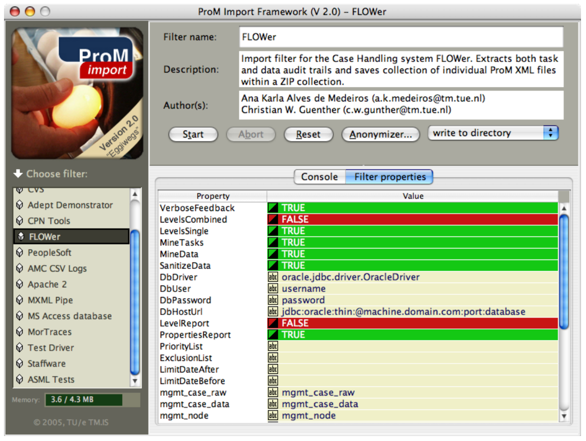
As this concept implies that all audit trail entries of an import session have to be buffered, before the first of them can be written, the process instance buffers are implemented to swap their content partially to disk storage.

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 [14] 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 [16] 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. Notice 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



Filter name: FLOWer

Description: Import filter for the Case Handling system FLOWer. Extracts both task and data audit trails and saves collection of individual ProM XML files within a ZIP collection.

Author(s): Ana Karla Alves de Medeiros (a.k.medeiros@tm.tue.nl)
Christian W. Guenther (c.w.gunther@tm.tue.nl)

write to directory

Property	Value
VerboseFeedback	TRUE
LevelsCombined	FALSE
LevelsSingle	TRUE
MineTasks	TRUE
MineData	TRUE
SanitizeData	TRUE
DbDriver	oracle.jdbc.driver.OracleDriver
DbUser	username
DbPassword	password
DbHostUrl	jdbc:oracle:thin:@machine.domain.com:port:database
LevelReport	FALSE
PropertiesReport	TRUE
PriorityList	
ExclusionList	
LimitDateAfter	
LimitDateBefore	
mgmt_case_raw	mgmt_case_raw
mgmt_case_data	mgmt_case_data
mgmt_node	mgmt_node

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, wich 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 an utmost 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 a 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 [17], 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.

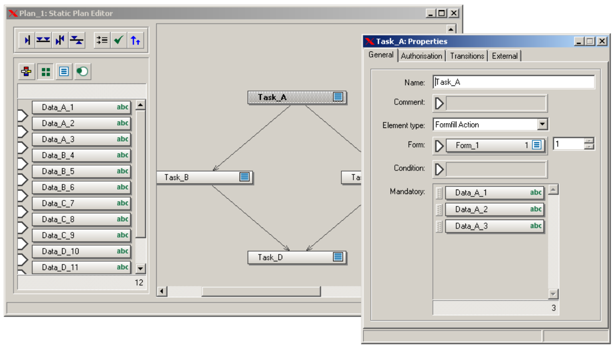
3. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
4. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process 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. A.K. Alves de Medeiros and C.W. Günther. Process mining: Using cpn tools to create test logs for mining algorithms. In K.Jensen, editor, *Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005)*, volume 576 of *DAIMI*, Aarhus, Denmark, October 2005. University of Aarhus.
10. 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.
11. 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.
12. 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.
13. 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.
14. C.A.R. Hoare. Algorithm 64: Quicksort. *Commun. ACM*, 4(7):321, 1961.
15. 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.
16. D.E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison Wesley, Reading, MA, USA, 2 edition, 1998.
17. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

A Import Plug-ins [3]

A.1 FLOW*er*

The case handling system FLOW*er* is an excellent object for process mining research, due to its feature set which greatly exceeds the capabilities of traditional workflow management systems. The subsequent sections introduce FLOW*er*, give an overview about the event log data available and its location, and sketch the conversion process implemented in the respective import plug-in.

Introduction Case handling systems are of particular interest for process mining research. Due to their high flexibility in process enactment, the resulting event logs exhibit a large degree of variation between process instances. Further, the fact that case handling systems are inherently data-driven provides the opportunity to also obtain event logs on a data modification level.



Name: Task_A
Element type: Formfill Action
Form: Form_1 1
Mandatory: Data_A_1, Data_A_2, Data_A_3

Fig. 6. Process designer component of FLOW*er*

The process designer component of FLOW*er* is depicted in Figure 6. For every task three distinct roles can be defined, determining the set of users who can *"execute"*, *"skip"* and *"redo"* this task. The first role allows traditional usage, while the second enables authorized users to bypass this task, thereby providing an elegant way to model exceptional behavior. Finally, users who have been granted the "redo" role can roll back this task, by which the implicit modeling of loops and iterations becomes feasible.

Figure 7 shows the end user interface of FLOW*er*. The so-called "wavefront", visible in the window on the upper right, indicates the status of tasks within a process instance and presents them to the user. This status is not defined explicitly, like committing a task in a workflow management system, but determined

[3] We would like to point out that we do not intend to include this appendix in the final workshop contribution. However, we feel that the additional information given here may be a useful reference for the editors.

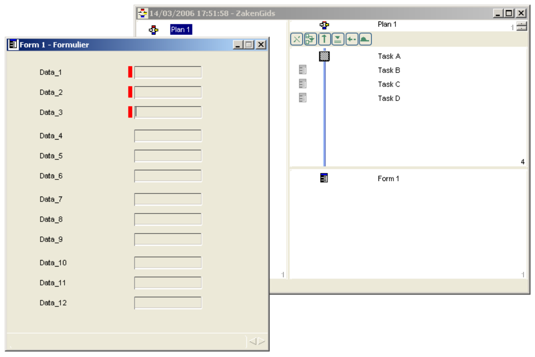


Fig. 7. End user interface of FLOW*er* (wavefront and form)

by the system. For each task, a set of data objects has to be defined which have to be set before the task is considered to be finished. This concept provides a great share of flexibility in case handling systems, as boundaries between tasks become less meaningful.

Data Source FLOW*er* provides event log information on two distinct levels. Next to recording the life-cycle changes of tasks in a process instance, the system also provides fine-grained events referring to changes made to single data objects. It is worthwhile to mention that events in the latter kind of logs are recorded in a very precise manner, i.e. as soon as an input field on a form has been confirmed a respective event is recorded by the system.

Audit trails are not recorded by default in FLOW*er*, the process designer rather needs to explicitly enable logging for each process definition. In this, the recording of events can be turned on and off in a fine-grained manner for each single task, which allows limit the amount of log data created.

Regarding the storage of event log data, FLOW*er* supports both recording audit trails to a database and writing a comparable set of information to plain text files, while both options can be enabled independently. The import filter plug-in for FLOW*er* does, at the time of writing, only support the conversion of audit trail information kept in a database.

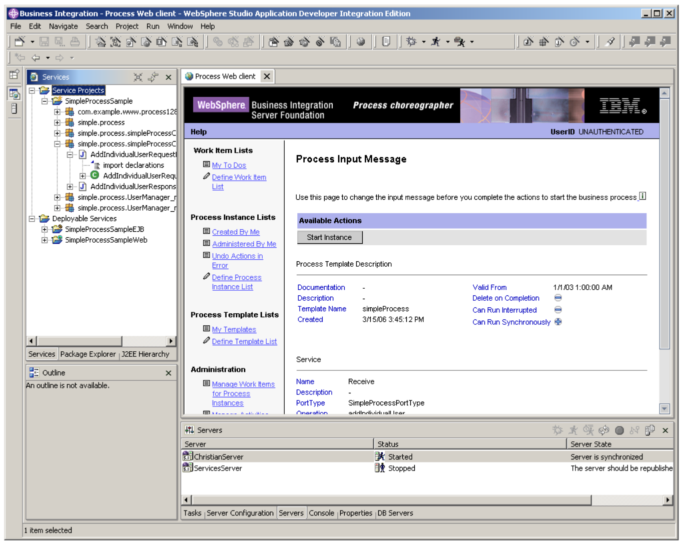
Conversion The plug-in implementation directly connects to the database where the event logs are stored. While we cannot disclose the concise structure of the FLOW*er* database, it can be said that accessing the information necessary requires multiple cascaded SQL queries, involving joins among several tables, for each event. Part of the logic which is assembling the necessary data is performed on the database server, by interpreting the SQL queries, while a decent amount of pre- and post-processing is implemented in the plug-in itself.

FLOW*er* is among the systems for which the most advanced support exists within the framework. Thus the import plug-in provides an extensive set of options for pre-processing the information, tuning the plug-in to the concrete FLOW*er* installation at hand, and selecting the subset of information to be recorded.

Logs on the task level and data modification level are exported separately, and can even optionally be split up with respect to the layers of abstraction defined in the FLOW*er* process definition.

A.2 WebSphere Process Choreographer

The Process Choreographer is part of IBM's WebSphere suite, providing a means to implement high-level business processes based on the BPEL language. In the last years, the paradigm of a service-oriented architecture (SOA), in which the orchestration of web services via BPEL plays a vital role, has increasingly gained in importance. Thus, WebSphere, as one of the implementations with the strongest industry backing, is an obvious and promising source of event logs.


Business Integration - Process Web client - WebSphere Studio Application Developer Integration Edition
File Edit Navigate Search Project Run Window Help
Services
Service Projects
SimpleProcessSample
com.example.www.process128
simple.process
simple.process.simpleProcessC
simple.process.simpleProcessC
AddIndividualUserRequest
import declarations
AddIndividualUserRequ
AddIndividualUserRespons
simple.process.UserManager_r
simple.process.UserManager_r
Deployable Services
SimpleProcessSampleEJB
SimpleProcessSampleWeb
Services Package Explorer J2EE Hierarchy
Outline
An outline is not available.
Process Web client
WebSphere Business Integration Server Foundation
Process choreographer
IBM
Help
UserID UNAUTHENTICATED
Work Item Lists
My To Dos
Define Work Item List
Process Instance Lists
Created By Me
Administered By Me
Undo Actions in Error
Define Process Instance List
Process Template Lists
My Templates
Define Template List
Administration
Manage Work Items for Process Instances
Available Actions
Start Instance
Servers
Tasks Server Configuration Servers Console Properties DB Servers
1 item selected

Process Input Message

Use this page to change the input message before you complete the actions to start the business process

Process Template Description

Documentation	-	Valid From	1/1/03 1:00:00 AM
Description	-	Delete on Completion	⊖
Template Name	simpleProcess	Can Run Interrupted	⊖
Created	3/15/06 3:45:12 PM	Can Run Synchronously	⊕

Service

Name	Receive
Description	-
PortType	SimpleProcessPortType

Server	Status	Server State
ChristianServer	Started	Server is synchronized
ServicesServer	Stopped	The server should be republishe

Fig. 8. Process designer component of WebSphere

Introduction The BPEL language allows for the specification of business processes which are essentially composed of web services, which serve as implementation for the defined tasks. As the interface represented by a web service is very generic, any application or system can, at least partially, be wrapped and presented in the form of one or multiple web services.

This paradigm allows for building heterogeneous and distributed applications in the form of business processes, which can again be exposed as web services to be used in further settings. In this way, BPEL allows for the successive assembly of high-level business logic from previously disconnected components, including legacy systems.

IBM provides an environment for implementing BPEL-based business processes with the Process Choreographer, as part of their WebSphere suite.It integrates a BPEL enactment engine with a Java 2 Platform, Enterprise Edition (J2EE) application server, thus creating a very flexible and extensible platform.

The process designer component of WebSphere Process Choreographer is shown in Figure 8. It is based on the popular Eclipse SDK, and provides means for e.g. graphically designing BPEL process definitions, developing J2EE components, and testing process definitions on a built-in test server.

Data Source In WebSphere Process Choreographer, just as in FLOW*er*, the recording of audit trail information has to be explicitly enabled for each business process. The resulting information is stored in a database audit-log table. WebSphere Process Choreographer records events related to state transitions of process instances, basic activities, staff activities (i.e. tasks performed by human resources), invoke activities, and Java snippets. These events are based on defined state-transition models and allow the concise traceability of process executions in the system.

In addition to storing audit trail events in a database, WebSphere also supports the so-called “Common Event Infrastructure” (CEI). This infrastructure decouples event emitters from event consumers by an intermediary event server, which disperses events to registered listeners in an asynchronous manner.

Conversion The current implementation of the import plug-in for the WebSphere Process Choreographer is limited to extracting audit trail events from the respective database table. These events are straightforward to access, however their interpretation requires a number of mappings to MXML. For example, as the state-transition model of WebSphere differs from that of MXML, event types have to be mapped, which is in some cases not trivial.

In order to be able to limit the overwhelming set of information that WebSphere Process Choreographer provides in its event logs, the import plug-in features a number of configuration options for that end. Further, the database connection settings can be adjusted to the concrete installation of WebSphere at hand.

A.3 Staffware

Staffware is a typical example of a traditional workflow management system. Its comprehensive market penetration renders this system an import candidate for process event log conversion.

Introduction Staffware is a workflow management system in the traditional sense, i.e. it supports the distributed execution of rigidly predefined process definitions. Although it is a rather inflexible system when it comes to dynamically adapting or evolving process definitions, its high performance in process execution makes it the preferred choice for many organizations.

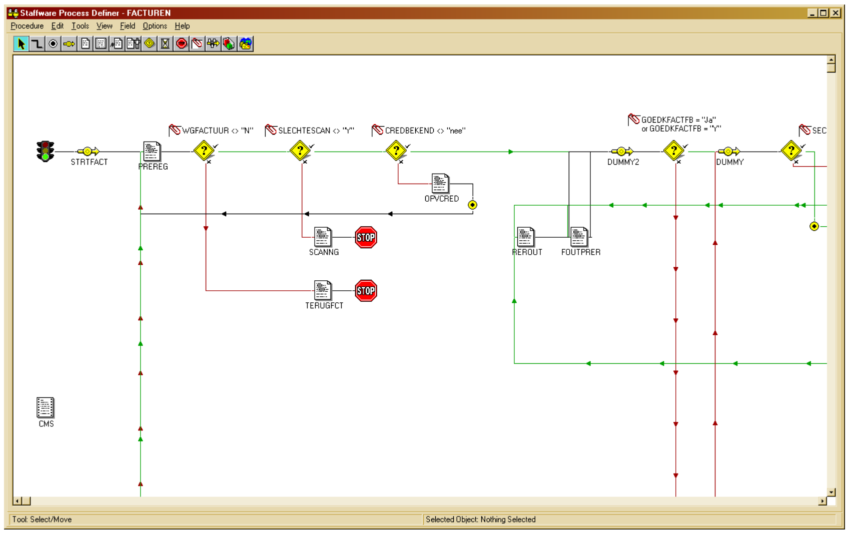

Staffware Process Definer - FACTUREN
Procedure Edit Tools View Field Options Help
WGFACTUUR <> "N"
SLECHTESCAN <> "Y"
CREDBEKEND <> "nee"
GOEDKFACTFB = "Ja" or GOEDKFACTFB = "Y"
SEC
STRTFACT
PREREG
OPVCRED
DUMMY2
DUMMY
SCANNG
TERUGFCT
STOP
STOP
REROUT
FOUTPRER
CMS
Tool: Select/Move
Selected Object: Nothing Selected

Fig. 9. Process designer component of Staffware

Figure 9 shows the process designer component of Staffware. For defining processes, Staffware uses a custom modeling language, which is burdened with a number of ambiguous and confusing properties.

```
Case 2
Diractive Description    Event            User             yyyy/mm/dd hh:mm
------------------------------------------------------------------------------
                         Start            rick@staffw_e    2005/04/11 17:09
TASKA                    Processed To     rick@staffw_e    2005/04/11 17:09
TASKA                    Released By      rick@staffw_e    2005/04/11 17:09
TASKB                    Processed To     martin@staffw_e  2005/04/11 17:09
TASKB                    Released By      martin@staffw_e  2005/04/11 17:10
TASKC                    Processed To     rick@staffw_e    2005/04/11 17:10
TASKC                    Released By      rick@staffw_e    2005/04/11 17:10
TASKE                    Processed To     tania@staffw_e   2005/04/11 17:10
TASKE                    Released By      tania@staffw_e   2005/04/11 17:11
                         Terminated                        2005/04/11 17:11
```

Logfile 1: Excerpt of a Staffware event log file.

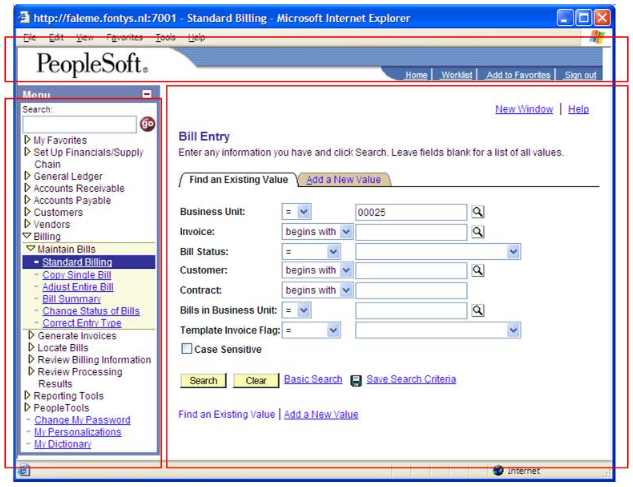
Data Source In the Staffware system, process event logs are exported in plain text, while for every process instance one log file is created. The files are structured in a human-readable, tabular manner. An example of a log file for one process instance is shown in Logfile 1.

Conversion The import plug-in for Staffware takes as input a collection of log files, or alternatively a folder which is supposed to contain such files. These files are subsequently parsed, while the first line provides the name of the process instance (which is "Case 2" in the excerpt shown in Logfile 1). The second and third line are skipped, as they always contain the same table headings.

The rest of the file features information about the events having occurred in that process instance, one event per line. These events are converted to audit trail entries, buffered and sorted, and finally written to the log writing pipeline.

A.4 PeopleSoft Financials

PeopleSoft Financials is a module of the Enterprise Resource Planning (ERP) system PeopleSoft, which can be used for the financial administration of an organization.



Bill Entry

Enter any information you have and click Search. Leave fields blank for a list of all values.

Business Unit:	=	00025
Invoice:	begins with	
Bill Status:	=	
Customer:	begins with	
Contract:	begins with	
Bills in Business Unit:	=	
Template Invoice Flag:	=	

☐ Case Sensitive

Fig. 10. User interface of PeopleSoft Financials

Introduction The PeopleSoft system consists of an application server, which is backed by a database system. This application server uses a web-based user interface, which is shown in Figure 10 to communicate with the users.

High-level tasks within the financial domain are represented by sub-modules, which can be selected on the left pane of the user interface. This will trigger the application server to display a set of pages on the right display pane, between which the user can navigate back and forth. These pages contain input fields and controls on web forms, and allow the user to enter the requested data into the system.

```
PSAPPSRV.2396    1-22265  16.47.13    0.000 Cur#2.FFAC RC=0 Dur=0.000
        Connect=FFAC/SYSADM/
PSAPPSRV.2396    1-22266  16.47.13    0.000 Cur#2.FFAC RC=0 Dur=0.000
        COM Stmt=SELECT COUNT(*) FROM PSMSGNODEDEFN
PSAPPSRV.2396    1-22267  16.47.13    0.000 Cur#2.FFAC RC=0 Dur=0.000
        COM Stmt=SELECT MSGNODENAME FROM PSMSGNODEDEFN ORDER BY MSGNODENAME
PSAPPSRV.2396    1-22268  16.47.13    0.000 Cur#2.FFAC RC=0 Dur=0.000
        Disconnect
PSAPPSRV.2396    1-22269  16.47.13    0.031 Cur#1.FFAC RC=0 Dur=0.000
        COM Stmt=select PT_CTI_AGENTID, PT_CTI_QUEUE, PT_CTI_CONFIGID from PS_PT_CTI_AGENT A
        where OPRID = :1 AND EFFDT = (SELECT MAX(EFFDT) FROM PS_PT_CTI_AGENT B WHERE
        A.OPRID= B.OPRID AND EFFDT <= TO_DATE(TO_CHAR(SYSDATE,'YYYY-MM-DD'),'YYYY-MM-DD'))
PSAPPSRV.2396    1-22270  16.47.13    0.000 Cur#1.FFAC RC=0 Dur=0.000
        Bind-1 type=2 length=6 value=875273
PSAPPSRV.2396    1-22271  16.47.13    0.000 Cur#2.FFAC RC=0 Dur=0.000
        Connect=FFAC/SYSADM/
PSAPPSRV.2396    1-22272  16.47.13    0.000 Cur#2.FFAC RC=0 Dur=0.000
        COM Stmt=SELECT MCFUQAGENTID, MCFREN_LOG_RTENT, MCFREN_PHYS_RTENT, MCFUQQUEUEID,
        MCFUQSKILLLEVEL, MCFUQMAXWKLD FROM PSMCFUQAGENTQ  WHERE MCFUQAGENTID = :1
PSAPPSRV.2396    1-22273  16.47.13    0.000
        Cur#2.FFAC RC=0 Dur=0.000 Bind-1 type=2 length=6 value=875273
PSAPPSRV.2396    1-22274  16.47.13    0.000
        Cur#2.FFAC RC=0 Dur=0.000 Disconnect
```

Logfile 2: Excerpt of a PeopleSoft Financials trace file.

Data Source In the PeopleSoft system, it is possible to enable a very fine-grained and verbose kind of logging for certain modules or sub-modules. Logfile 2 shows an excerpt of such a PeopleSoft log. As these logs are very detailed, it is not practicable to record them over an extended period of time.

One can see that, among further information, this log contains SQL queries. These queries have been used by the application server to request data for constructing the single pages of a sub-module.

Conversion The import filter plug-in for PeopleSoft analyzes these low-level logs for commands which correspond to the creation of pages. Every creation of a page is considered an event, and stored in audit trial entries accordingly. Subsequently, it extracts from the corresponding SQL query the set of primary keys used for assembling each respective page. By comparing the sets of primary keys used for each page, the plug-in heuristically tries to group audit trail entries of pages, which are supposed to have been part of one sequence, i.e. sub-module.

The described import filter implementation is the most successful of three different attempts to extract logs from the PeopleSoft Financials system. However, its results can not be used for productive process mining analysis, as they are not sufficiently reliable.

A.5 CPN Tools Simulation

Apart from its outstanding modeling and analysis support for Colored Petri Nets (CPNs), CPN Tools provides the means to execute and simulate models. With the help of an extension, the simulation engine of CPN Tools can be used to create synthetic test logs for research purposes.

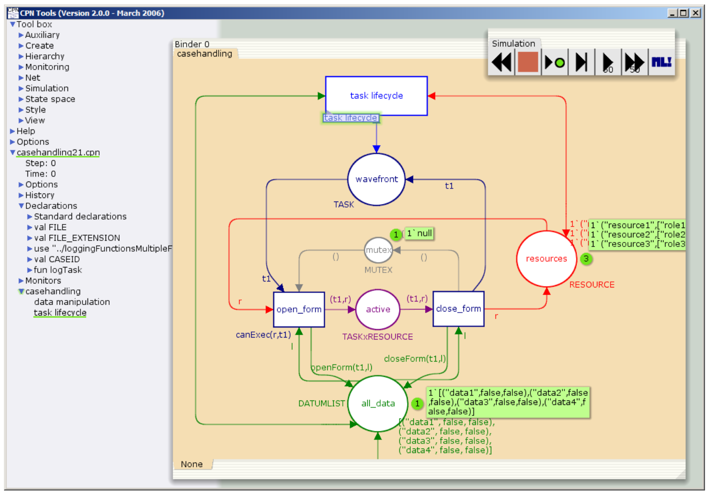


Fig. 11. CPN Tools

Introduction CPNs are a high-level variant of Petri nets, which are enhanced with structured data definitions (i.e "colors") for tokens and places. In addition, transitions and arc inscriptions can be enhanced with expressions in the Standard ML programming language. These features make CPN a versatile modeling language, which can be employed in a number of different applications.

The application CPN Tools provides excellent support for modeling CPN models. Further, it has extensive features that allow for formal correctness checking, and for the simulated execution of models.

Log File Generation In order to exploit the powerful modeling and simulation capabilities of CPN Tools for process mining research, an extension to this application was developed in Standard ML. This extension can be used within models, providing functions for writing audit trail entries e.g. when particular transitions have been fired.

The output of simulating a CPN model which was enhanced with these logging functions is a set of text files. Each of these files includes all audit trail entries for one process instance, formatted in MXML.

After simulating a CPN model, these files can be assembled to standard-compliant MXML files using the CPN Tools import filter plug-in in the framework. The implementation of this plug-in is rather simple, as it only concatenates the supplied input files, adding proper environments for process definitions and process instances.

Although rather simple, this feature has proven to be very effective for research purposes. It can be employed for rapid prototyping of test logs, simply by quickly modeling and simulating a respective example process in CPN Tools. On the other hand, existing models can be extended with logging capabilities in a straightforward manner, which allows for their subsequent analysis using process mining techniques.

For an extensive description of the CPN Tools logging extensions, the reader is referred to [9].

A.6 CVS

The "Concurrent Versioning System" (CVS) is a source code management (SCM) system, and as such does not represent a process-aware information system in the traditional sense. However, as the software development process is well reflected in the sequence of commit operations to a source code repository, these systems are interesting objects of research for several process mining techniques.

Introduction Of all SCM systems on the market, CVS is certainly one of those which have been around for the longest time. This fact, however, does not seem to influence its active usage in any negative way. CVS is still the de-facto standard for SCM solutions in research, industry, and most notably open source projects around the world.

This fact is most likely strongly due to excellent and widespread tool support for CVS, with client software being integrated in almost every development environment. The system itself consists of a server component, which manages repositories of files and servers them to authenticated users over the network.

Data Source The CVS server does not store a log of repository operations in a dedicated file. However, valid users can export a so-called "change log" from the server, using the command "`cvs log`" in a terminal. This results in printing information summaries for commits to the repository to the standard output. An excerpt of such a change log is shown in Logfile 3.

Conversion Logfile 3 shows an excerpt of a CVS change log. The excerpt shows the commit history for one particular file in the repository, namely "splashos2.rc" (as referenced in the first two lines). After giving some general information about the artifact itself, the actual revision history starts after a separator line made of a sequence of "=" characters.

```
RCS file: /cvsroot/mozilla/mail/app/splashos2.rc,v
Working file: mozilla/mail/app/splashos2.rc
head: 1.4
branch:
locks: strict
access list:
symbolic names:
    THUNDERBIRD_1_5rc1_RELEASE: 1.4
    MOZILLA_1_8rc1_RELEASE: 1.4
    FIREFOX_1_5rc1_RELEASE: 1.4
[...]
keyword substitution: kv
total revisions: 4; selected revisions: 4
description:
----------------------------
revision 1.2
date: 2004/09/07 22:24:44;  author: darin%meer.net;  state: Exp;  lines: +0 -9
fixes bug 258217 "Windows DDE code should extract application name from nsXREAppData"
----------------------------
revision 1.1
date: 2003/11/25 18:25:07;  author: mkaply%us.ibm.com;  state: Exp;
RC file for Os/2 thunderbird
=============================================================================
```

Logfile 3: Excerpt of a CVS change log file.

The import plug-in implementation regards each combination of an artifact and an associated commit revision as an event. As can be seen in Logfile 3, revision entries are separated by lines containing only dashes ("-").

The challenge in converting CVS change logs to process event logs lies in the fact, that CVS orders its change logs by modified artifacts (i.e. source code files). However, the MXML format specifies that events be ordered in a strictly occurrence-based fashion (i.e. the order in the log should represent the order of occurrence).

Thus, all parsed events are buffered and aggregated before transmitting them to the log writing pipeline. The result is one global process and process instance, which contains an event for each committed revision for each versioned file in the repository.

A.7 Subversion

The Subversion SCM system has been developed to address a number of shortcomings in the CVS architecture. It is increasingly gaining supporters and users throughout the industry, and particularly with open source projects.

Introduction While the usage paradigm of Subversion is clearly based on CVS, its main goal is to address a number of shortcomings in CVS as perceived by a majority of users. The list of enhancements includes transaction support, an enhanced algorithm for determining the differences between two revisions of a versioned file (making the differentiation between binary and text-based files known from CVS obsolete), and many more.

Subversion is based on a repository, which can be accessed by a number of different servers, each using different communication protocols and authentication mechanisms. There exists both a command line client, a number of graphical standalone clients, and clients integrated in development environments for all major platforms.

```
------------------------------------------------------------------------
r95 | christian | 2006-03-06 20:14:37 +0100 (Mon, 06 Mar 2006) | 3 lines
Changed paths:
   M /PromImport/.classpath
   M /PromImport/build.xml
   A /PromImport/src/org/processmining/importfilter/filters/TestDriver.java
   M /PromImport/src/org/processmining/importfilter/gui/ImportFilterFrame.java
   A /PromImport/src/org/processmining/importfilter/gui/MemoryGauge.java
   M /PromImport/src/org/processmining/importfilter/sorting/ATESortingProxy.java
   M /PromImport/src/org/processmining/importfilter/sorting/SortedATEDiskBuffer.java
   M /PromImport/src/org/processmining/importfilter/sorting/SortedEventLogs.java
   A /PromImport/src/org/processmining/importfilter/util/LogWriterDevNull.java

Fully integrated the sorting infrastructure into the framework.
Added dummy filter for generating audit trail entries and dummy sink (logwriter) for
discarding output of the pipeline.
Interface is enhanced with memory gauge below the filter list.
------------------------------------------------------------------------
r94 | akamedeiros | 2006-03-06 13:49:19 +0100 (Mon, 06 Mar 2006) | 1 line
Changed paths:
   M /PromImport/src/org/processmining/importfilter/filters/EastmanWorkflow.java

EastmanWorkflow.java
```

Logfile 4: Excerpt of a Subversion change log file.

Data Source Similar to CVS, the change log of a Subversion repository can be exported with the command "`svn log <repository_url>`" from a terminal. An example excerpt of the resulting output is shown in Logfile 4.

Conversion CVS works on a artifact basis, which is why it lists the history of the repository per-file in change logs. Subversion, on the other hand, supports transactions, which clearly shows in its change logs. Every commit to the repository is atomic, can affect any subset of the artifacts versioned in the repository, and is equipped with a unique revision number.

Thus, the change log is organized as follows. The first line of each commit entry gives the revision number, the user having committed this revision, and the exact date and time of the commit. This is followed by a list of artifacts modified by this commit revision. A line starting with "M" indicates that the artifact has been modified, while lines starting with "A" indicate artifacts which have been added to the repository with this commit.

This list of affected files is followed by a blank line, after which the user comment given for this commit is printed. Commit revision entries in the change log are separated by lines featuring only a sequence of the "=" character (i.e. Logfile 4 shows two revisions: Revision 94 and 95).

As the change log is ordered backwards (i.e. the latest revision is featured at the beginning of the change log file), events need to be buffered and re-ordered before passing them to the log writing pipeline. For each commit revision parsed, one event is created for every artifact affected (i.e. Revision 95 in Logfile 4 would lead to nine events in the resulting MXML log).

A.8 Apache

Web server access logs are not a typical object for process mining research. However, depending on their interpretation the application of process mining algorithms can reveal interesting information. Moreover, the widespread availability of web server access logs makes them ideal for anybody just wanting to play around with process mining on their very own log files.

Introduction The Apache foundation develops and maintains the most successful web server to date, released free of charge under an open source license. It is available for all major platforms and operating systems, and it can be customized and extended in numerous ways, using modules and scripts which are freely available on the web.

```
129.49.77.149 - - [03/Jul/2005:01:43:03 +0200] "GET /research/patterns/_themes/modular/amodrule.gif HTTP/1.1"
        200 200 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
129.49.77.149 - - [03/Jul/2005:01:43:10 +0200] "GET /research/patterns/_derived/standards.htm_cmp_modular110_vbtn.gif HTTP/1.1"
        200 452 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
207.46.98.76 - - [03/Jul/2005:01:43:13 +0200] "GET /staff/anorta/XRL/bottom.htm HTTP/1.0"
        200 685 "-" "msnbot/1.0 (+http://search.msn.com/msnbot.htm)"
129.49.77.149 - - [03/Jul/2005:01:43:17 +0200] "GET /research/patterns/_derived/products.htm_cmp_modular110_vbtn.gif HTTP/1.1"
        200 443 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
129.49.77.149 - - [03/Jul/2005:01:43:17 +0200] "GET /research/patterns/_derived/patterns.htm_cmp_modular110_vbtn.gif HTTP/1.1"
        200 372 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
129.49.77.149 - - [03/Jul/2005:01:43:24 +0200] "GET /research/patterns/_derived/yawl.htm_cmp_modular110_vbtn.gif HTTP/1.1"
        200 361 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
129.49.77.149 - - [03/Jul/2005:01:43:30 +0200] "GET /research/patterns/_derived/wscl.htm_cmp_modular110_vbtn.gif HTTP/1.1"
        200 361 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
129.49.77.149 - - [03/Jul/2005:01:43:31 +0200] "GET /research/patterns/_derived/what's_new.htm_cmp_modular110_vbtn.gif HTTP/1.1"
        200 410 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
192.91.147.35 - - [03/Jul/2005:01:43:33 +0200] "GET /research/patterns/_derived/home_cmp_modular110_home_a.gif HTTP/1.0"
        304 - "-" "Mozilla/4.0 (compatible;)"
129.49.77.149 - - [03/Jul/2005:01:43:37 +0200] "GET /research/patterns/_derived/links.htm_cmp_modular110_vbtn.gif HTTP/1.1"
        200 355 "http://is.tm.tue.nl/research/patterns/" "Opera/7.23 (Windows NT 5.1; U)  [en]"
```

Logfile 5: Excerpt of an Apache 2 access log file.

Data Source Apache logs every request for a file, usually issued by web browser software, in so-called "access logs". An example excerpt of an access log is shown in Logfile 5: Every line in these logs represents one request, including the IP address and browser software of the requesting party, HTTP status codes, the exact time and date of the request, and the resource which was requested.

The format of these access logs can be altered by the administrator, which makes web server logs potentially non-standardized. However, most administrators choose to use the standard logging configuration specified in the configuration file shipped with Apache.

The exact location of web server logs differs between operating systems, and can be changed at will by the administrator. However, typical locations include "/var/log/httpd" (Debian GNU/Linux and Mac OS X) and "/var/www/logs" (when running in a chroot on OpenBSD).

Conversion In its standard configuration, the import filter plug-in for Apache assumes that all recorded events belong to a single process instance of the same process definition. From each line of the access log exactly one audit trail entry is parsed, which has the IP address (or host name, if available) of the requesting party in the originator field. The workflow model element is made up by the requested file, while the time stamp is merely converted to fit the MXML standard.

A.9 MXML Pipe

Due to its modular and extensible log writing pipeline design, the ProM Import Framework is able to offer additional functionality, exceeding simple import and writing of event logs. The MXML Pipe plug-in makes this functionality available also to event log data which is already stored in the MXML format.

There are sometimes occasions, where one has imported a large amount of event logs from a system installation, but one has forgotten to anonymize this data appropriately. The MXML Pipe makes re-importing the affected data superfluous, as one can simply re-pipe the already imported MXML logs through the framework with the anonymizer settings adjusted accordingly.

The functionality of the MXML Pipe is rather straightforward to explain: Event logs are read from a file, parsed and transmitted into the log writing pipeline. This makes the event log data subject to the settings applied to the log writing pipeline once more, allowing users to tweak data long after the actual import session.