Mining Reference Process Models and their Configurations

Florian Gottschalk, Wil M.P. van der Aalst, Monique H. Jansen-Vullers

Eindhoven University of Technology, The Netherlands.
{f.gottschalk,w.m.p.v.d.aalst,m.h.jansen-vullers}@tue.nl

Abstract. Reference process models are templates for common processes run by many corporations. However, the individual needs among organizations on the execution of these processes usually vary. A process model can address these variations through control-flow choices. Thus, it can integrate the different process variants into one model. Through configuration parameters, a configurable reference models enables corporations to derive their individual process variant from such an integrated model. While this simplifies the adaptation process for the reference model user, the construction of a configurable model integrating several process variants is far more complex than the creation of a traditional reference model depicting a single best-practice variant. In this paper we therefore recommend the use of process mining techniques on log files of existing, well-running IT systems to help the reference model provider in creating such integrated process models. Afterwards, the same log files are used to derive suggestions for common configurations that can serve as starting points for individual configurations.

1 Introduction

Many supporting processes like, e.g., procurement or invoicing processes are organized similarly among companies. Reference process models depict such processes in a general manner, thus providing templates for individual implementations of these processes, and are available for various domains [8, 11, 16]. However, even supporting processes are rarely executed in exactly the same manner among companies. For that reason the corresponding reference models must be adapted to individual needs during the process implementation.

To support such an adaptation of reference models, suggestions for configurable process models have been made by various researchers (e.g. [3, 10, 13]). Configurable process models require that the reference process model provider combines different process variants into an integrated process model from which reference model users can then derive individual model variants by setting configuration parameters. In this way, each organization can derive the individual process model fitting their needs while this derived process model remains conform with the used reference model. That means, the adaptation process does not require a manual process modeling which always comes with the risk of error and which could thus jeopardize a process's executability.

While this simplifies the adaptation process for the reference model user, it is on the expense of the model provider who has to construct a more complex model integrating several process variants. As handling the complexity and avoiding errors is already difficult for traditional reference models [6], we focus in this paper on providing support for the creation of configurable process models.

Usually, the processes represented by a reference model are not new and innovative approaches, but rather derived from established process variants. That means, when a configurable reference model is built, various variants of the process are already in place in organizations. In traditional reference model creation approaches, process analysts are examining the processes in place, e.g. through expert and user interviews, and then compile the gained information "in their minds" to a best-practice reference model while they abstract from specific requirements of individual organizations. A configurable model, however, should already provide everything that is necessary to derive a model satisfying the specific requirements. For that reason, it needs to include these specific aspects of different process variants. Thus, an abstraction is not needed and the model should rather be based on what is really happening.

Extensive protocols about what has happened are usually widely available in today's IT environments in the form of so-called log files. Data and process mining techniques have been developed to gain condensed information about the process behavior from such log files. In the following we will depict how such existing techniques can directly support the creation of a configurable reference model by using the log files from various IT systems.

Figure 1 provides an overview of the suggested approach. At first, the available log files from various systems must be prepared through filtering of irrelevant content and mapping of different naming conventions among different log files. This is subject of Section 2. Afterwards, process mining techniques can be used not only to create process models for individual systems but also to build process models which are valid for all the systems, i.e. which integrate the various process variants. We show this in Section 3. Section 4 depicts how such integrated models covering multiple variants of a process can be configured to derive the individual variants before the paper ends with some conclusions.

2 Pre-processing the log files

Many of today's IT systems constantly write log files to record functions that are executed in the system, changes that are made to the system or its data, "system alive" status updates and so on. For example, most web-servers write an entry into a log file for each single requested page including information about the time of access, the IP address of the user, whether the access was successful, and maybe even a user name or submitted data. Due to today's extensive use of information systems in all business areas, such log files containing information about the executed business processes are usually widely available.

In this paper we focus on deriving the control flow among the different activities of a business process from such log files. For this purpose many details of log

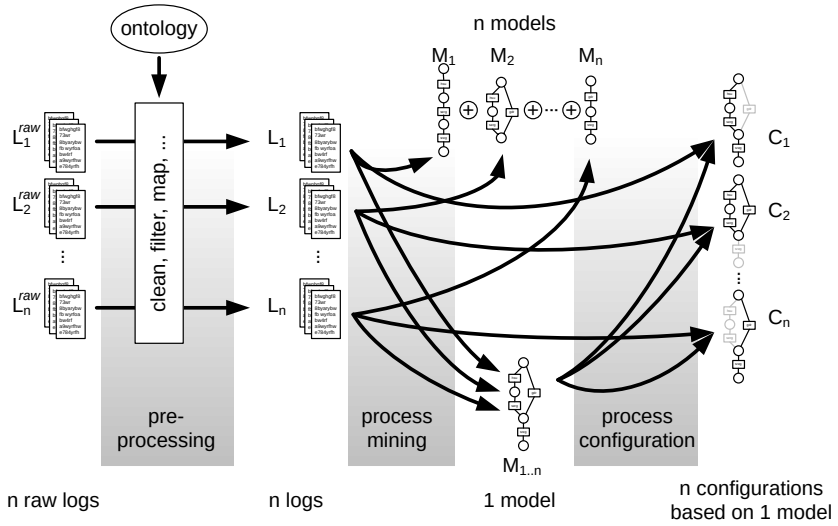

Fig. 1. Deriving integrated process models and their configurations from log files

files are irrelevant and it is here sufficient if we consider a log file as being a set of event traces. Each event trace is then an ordered set of the activity identifiers classifying each log event as the execution of the particular activity.

Definition 1 (Log file). *Let E be a set of activity identifiers. Then*

- $I = E^*$ *is the set of all possible event traces, i.e.* $\langle e_1, ..., e_n \rangle \in I$
- $events : I \rightarrow I\!P(E)$ *is a function defined such that* $events(\langle e_1, ..., e_n \rangle) = \{e_i | 1 \leq i \leq n\}$ *is the set of all activity identifiers in an event trace* $\langle e_1, ..., e_n \rangle$,
- $L \subseteq I$ *is a log file, i.e. a set of event traces, and*
- $\Gamma = I\!P(I)$ *is the set of all such log files.*

For generating reference models, it is very important to gather log files from various systems executing the process in question. The selection of sites depends on the purpose of the model that should be created. If a model should represent configuration options of a software that is distributed internationally, various sites running successful implementations of the software should be chosen from different countries. If a model should represent good examples for a certain process, various successful implementations of that process should be chosen, but these do not necessarily need to be implemented using the same software. All in all, *the source of the used log files should widely cover the targeted scope and all aspects of the model which should be created.* Let us thus in the following assume that a comprehensive set $L^{raw} = \{L_i^{raw} | 1 \leq i \leq n\}$ of n such raw input log files is available (see the far-left of Figure 1).

Although log files are widely available today, the purpose of their creation and their level of details varies. While the transaction management of databases

requires very explicit and detailed logs for being able to undo all changes completely automatically, sometimes log entries are rather unspecific debug messages introduced and never removed by the programmer to find errors in the code. In any way, the log files are rarely created for deriving process models.

For that reason, the available log files must be pre-processed before we can use actual mining techniques to discover meaningful behavioral patterns, i.e. the log files have to be cleaned of irrelevant data and relevant data has to be aligned. When building configurable reference process models, three aspects are especially important in this phase.

- At first, the data in the log files has to be anonymized. Log files usually contain a lot of personal data. This information is highly confidential and the usage is in most cultures strongly restricted through privacy rights and laws. As configurable reference models target at the re-use by others, it is especially important that no personal information is retained in the model. Hence, the elimination of such personal information should take place before any data is processed.
- Secondly, the level of details of the log files has to be balanced among the different input log files and adjusted to the level targeted for the resulting model by aggregating related log events. Otherwise, the level of details in the generated process model will later on be highly inconsistent among different process branches. To reach this balanced level of details an ontology can for example be used. Then single or groups of log events can be mapped onto an agreed level of ontology classes.
- As the same ontological concept is hardly called in the same way by different sources, it must also be ensured that log events from the different source log files are mapped onto each other. This might already come with the use of a common ontology for adjusting the level of details. Otherwise, a direct matching of event names might also be possible.

Further details on how to perform such pre-processing steps can, e.g., be found in [4, 12, 17]. Process and data mining efforts — as also described in the remainder of this paper — heavily depend on the quality of the pre-processed log files. Therefore, pre-processing comprises in general 60–80 percent of the whole processing efforts [4, 17].

In the following we say that pre-processing is a function $prep : \Gamma \rightarrow \Gamma$ which performs all mentioned pre-processing steps for each log file, including a re-naming of log events belonging to the same ontology class to a common class name. The result of the pre-processing is then a consistent set of log files $L = prep(L^{raw})$.

3 Process Mining

The pre-processed log files can serve as the input for a process mining algorithm. Process mining has proven to be a valuable approach for gaining objective insights into business processes which are already in place in organizations. Such

algorithms search for re-occurring patterns in the execution traces of the system in question and generalize the overall process behavior as process models.

To depict process models we will use a workflow net representation here. While the process model types used by existing process mining algorithms vary, most of these models can be transformed into workflow nets using appropriate transformation algorithms as, e.g., provided by the process mining framework ProM [1, 7]. Workflow nets are a formal notation to represent business processes based on Petri nets. Within workflow nets, transitions depict the activities that should happen during the process execution. Through arcs transitions can be connected to places which then represent the conditions resulting from the execution of the transitions. Places also represent the pre-conditions for the execution of transitions whenever an arc connects a place to a transition. A special input place always represents the start of the process, while a special output place represents its completion.

Definition 2 (Workflow net). *A workflow net is a triple $M = (P, T, F)$, such that:*

- *P is a set of places,*
- *T is a set of transitions ($P \cap T = \varnothing$),*
- *$\mathbf{i} \in P$ is the unique input place,*
- *$\mathbf{o} \in P$ is the unique output place,*
- *$F \subseteq (P \setminus \{\mathbf{o}\} \times T) \cup (T \times P \setminus \{\mathbf{i}\})$ is a set of arcs (flow relation), and*
- *Δ is the set of all workflow nets*

Process mining helps process analysts to determine the processes executed by organizations either to document or to improve them. Although reference models are derived from well-running systems, this does not imply that these processes are documented by models, correctly describing the executed behavior. Thus, process mining can also help the reference model designer who has log files from successful process implementations available to create process models.

For this paper we define a mining algorithm as follows:

Definition 3 (Mining algorithm). *A mining algorithm α maps a log file onto a workflow net, i.e.*

$$\alpha : \Gamma \rightarrow \Delta.$$

We thus abstract from the particularities of the mining algorithm, i.e. α may be any process mining algorithm [1, 2]. Further on, we assume that the result of the algorithm fulfills the requirements of a workflow net. This is trivial to achieve for any algorithm that provides a Petri net (or a model that can be transformed into a Petri net) by connecting a unique input place to all its initial elements and a unique output place from all its final elements [5].

For each log file $L_i \in L$ used for the creation of a reference model, a process mining algorithm can therefore generate a process model $M_i = \alpha(L_i)$ depicting the behavior of the log file's original system (see top of Figure 1). Simplified[1],

[1] The description here provides a brief idea of what a process mining algorithm does. In practice process mining is far more complex as the algorithms, e.g., have to take concurrency, incomplete logs, noise, or invisible tasks into consideration [2].

a process mining algorithm splits a log file into the event traces of individual cases, i.e. process instances. It then constructs the process model by analyzing and comparing the events in the traces. Each log event is mapped onto a corresponding transition in the model. For each event that occurs in the event trace, the algorithms checks in the the so-far derived model if the corresponding transition can be reached from the transition corresponding to the preceding log event. Is this not the case, a choice is introduced at the place succeeding the transition corresponding to the preceding log event by adding a new arc leading from this place to the transition corresponding to the event in question. The resulting process model will thus depict that when reaching the particular point of the process, the process flow can either continue as all the previous traces did or it can continue as this deviating event trace did.

After having derived a process model M_i for each log file $L_i \in L$, the process model designer can compare these models with each other. By manually or automatically aligning and merging them, an integrated process model $M^+_{1..n} = M_1 \oplus M_2 \oplus ... \oplus M_n$ representing the behavior of all the individual models can be generated [9].

However, instead of deriving an individual process model for each system and later on trying to integrate these models, process mining algorithms can also be used to directly generate an integrated model for all of the log files in L. If we concatenate all the log files $L_i \in L$ into a single log file $L_{1..n} = \bigcup_{i=1..n} L_i$, the process mining algorithm α still works in exactly the same way on $L_{1..n}$ as it did for each of the individual log files. Due to the alignment of event names in the pre-processing, the algorithm is able to recognize which log events belong to the same class of events and match them. Thus, the algorithm just processes more process instances and creates a process model $M_{1..n} = \alpha(L_{1..n})$ that is valid for all these instances. That means, the resulting model usually contains more choices than each of the individual models M_i because a combined set of event traces might contain more variants than a subset of these traces. But, as this model represents the behavior of all the instances from the various systems, the model is in the same way an integrated process model valid for all the different input systems as a model generated from merging the individual models.[2]

Two properties are important when selecting a process mining algorithm for building such a process model integrating various process variants.

Among different systems it is well possible that steps executed in the process of one system are skipped in the other system. In such cases, the process mining algorithm must be able to introduce a by-pass for the skipped step in the generated process model, e.g. through adding a so-called invisible or silent transition as an alternative to the skipped transition. The invisible transitions then allow for state changes without corresponding to any log events and thus without representing any 'real' behavior.

[2] While the model created by merging several individually mined models should in theory represent the same behavior as the integrated model mined from the combined set of log files, the resulting models depend on the used process mining and merging algorithms and will thus hardly be identical in practice.

Further on, it might later on be desired that a process model can be derived from the reference model which does not conform exactly to the behavior of one of the systems used for the development of the reference model. Instead it might be necessary to combine various aspects of different systems which requires that the used process mining algorithm over-approximates the behavior of the input systems. Most process mining algorithms achieve this as they analyze choices between events only locally and neglect dependencies between choices that do not directly follow each other. By neglecting such non-local non-free choices, the resulting process models permit for example to chose in the beginning of the process a process part that only occurred in a subset of the process instances, while at a later stage a choice is made for a process part that was not part of any of these process instances.

An overview of process mining algorithms is provided in [2] while the ProM process mining framework [1, 7] provides implementations for many of these algorithms. The choice for a concrete algorithm and the quality of the resulting model very much depends on the input log files [15]. In controlled experiences with high-quality input data, we achieved good results using the multi-phase miner [5] because it guarantees the fitness of all the event traces to the resulting model. Using real-world data, it is however hardly possible to derive such high-quality log files during the pre-processing. In such cases algorithms that are able to deal with "noise" in the log files might perform better.

4 Deriving Configurations

The mined process model allows for the execution of all the process's various variants as it is based on the execution log files from the varying systems. Compared to a set of different models, the integrated model has the advantage for the process designer that later maintenance changes only need to be performed once on the integrated model, and not various times for each of the process variants. The integrated model also covers all combination possibilities of process parts which is usually impossible to achieve when providing a set of process variants. Naturally, reference model users do not need all these variants. Instead, they like to have a specific model covering the individually needed process behavior. Hence, the reference model user needs to configure the integrated model to that subset which depicts this desired behavior.

To define such a configuration for workflow nets, we simply say that a configuration is the set of all elements that should remain in the workflow net. In this way, the configured net can be derived by creating the intersections of the workflow net's elements with the configuration.

Definition 4 (Configuration). *Let $M = (P, T, F)$ be a workflow net. Then any $C \subseteq P \cup T \cup F$ such that $\{\mathbf{i}, \mathbf{o}\} \subseteq C$ and $\forall_{(n_1,n_2)\in F\cap C}\{n_1, n_2\} \subseteq C$ is a configuration of M. $M_C = (P \cap C, T \cap C, F \cap C)$ is the configured workflow net using configuration C.*

Of course, a careful selection must be made for the configuration as many configurations are not feasible, e.g. because they would eliminate the execution

of transitions that are essential for the process. For example, it is obviously impossible to check an invoice during an invoicing process, if the invoice has not been created beforehand. That means that in addition to the integrated model, the reference model provider also needs to offer some guidance to 'good' configurations.

Examples for such good configurations are the systems used to create the integrated process model. These established variants of the process could thus provide a better starting point for reference model users that want to derive their individual models than the complete integrated model of all process variants can be. If we know the configurations of the integrated model leading to the selected, established process variants, and if we know which of these variants might probably be the closest to our requirements (e.g. because of a comparable company size and target market) then the derivation of an individual process would normally just mean to slightly amend this given configuration by adding a number of elements from the integrated model to the configuration and/or removing some of them. In this way, the risky and time-consuming task of configuring the process from scratch can be avoided.

To determine such a configuration, we can re-use the (cleaned) log file of the particular system. It contains all behavior possible in the particular system and can be 're-played' on the integrated model. To depict how this re-play is performed, we first need to introduce the concept of a path of a workflow model.

Definition 5 (Path). *Let $M = (P, T, F)$ be a workflow model. Then $\Phi = \{\langle n_1, ..., n_m\rangle \in (P \cup T)^* | (\forall_{i=1..m-1}(n_i, n_{i+1}) \in F)\}$ is the set of paths of M. The set of elements of a path $\langle n_1, ..., n_m\rangle \in \Phi$ is defined by the function elements : $\Phi \rightarrow I\!P(P \cup T \cup F)$ such that elements$(\langle n_1, ..., n_m\rangle) = \{n_1, (n_1, n_2), n_2, (n_2, n_3), ..., (n_{m-1}, n_m), n_m\}$.*

To depict the re-play we assume that the integrated model was created by a mining algorithm like the simplified algorithm depicted in Section 3 which guarantees a fitness of 1, i.e. that the behavior of all traces of the log file L_i are represented by the integrated model, and that aspects like concurrency, noise, or incompleteness are neglected. In this way, the re-play starts for each trace of the log file from the input place **i** of the integrated model and searches for a path to a transition that corresponds to the first log event of the particular trace. This path should however not pass any visible transitions as their occurrence would require a corresponding event in the log file before the first event. Next, a path through places and invisible transitions is searched from this transition onwards to the next log event and so on. When the transition corresponding to the last log event of an event trace is found, the replay must conclude with finding a path from this last transition to the output place **o**. This process is repeated for every trace in the log file. The configuration of the model corresponding to the behavior of all these traces is then the set of all the transitions, places, and arcs used during the re-play. The individual model corresponding to this behavior can then be derived from the integrated model as depicted in Definition 4 and all unnecessary elements can automatically be dismissed.

Definition 6 (Log replay). *Let $M_{1..n} = (P, T_{vis} \cup T_{inv}, F)$ be a workflow net with T_{inv} as its set of invisible transitions, and let L_i be a log file. Moreover, let $\bigcup_{\theta \in L_i} events(\theta) \subseteq T_{vis}$ and $\Phi' = \{\langle n_1, ..., n_m \rangle \in \Phi | \{n_1, n_m\} \in T_{vis} \cup \{\mathbf{i}, \mathbf{o}\} \wedge \{n_2, ..., n_{m-1}\} \subseteq T_{inv} \cup P \setminus \{\mathbf{i}, \mathbf{o}\}\}$. Then*

$$\begin{aligned} C_i = & \bigcup \{elements(\langle \mathbf{i}, ..., e_1 \rangle) | \langle \mathbf{i}, ..., e_1 \rangle \in \Phi' \wedge \langle e_1, ... \rangle \in L_i\} \\ & \cup \bigcup \{elements(\langle e_k, ..., e_{k+1} \rangle) | \langle e_k, ..., e_{k+1} \rangle \in \Phi' \wedge \langle ..., e_k, e_{k+1}, ... \rangle \in L_i\} \\ & \cup \bigcup \{elements(\langle e_m, ..., \mathbf{o} \rangle) | \langle e_m, ..., \mathbf{o} \rangle \in \Phi' \wedge \langle ..., e_m \rangle \in L_i\} \end{aligned}$$

is the configuration of $M_{1..n}$ that corresponds to the behavior represented by L_i.

While the re-play of log files on models that were created using more sophisticated process mining algorithms is more complex (e.g. a log event might not have a corresponding transition or the path to the next log event might start from a different transition which corresponds to an earlier log event) [14], the configuration can still be discovered by simply identifying the visited model elements. An algorithm to perform such a complex re-play is for example part of the conformance checker provided by ProM.

5 Conclusions

In this paper we showed how reference process models can be constructed from log files of established business processes. Derived by proven process mining algorithms, these models depict and integrate the behavior of several different variants of a common business process. Such an integrated model can afterwards be restricted to the individual process variant required by an organization by means of configurations. By re-playing a log file of an existing system on the integrated model, a configuration of the model conforming to the system's behavior can be identified. Such configurations can serve as starting points for individual configurations. While the development of process mining and conformance checking methodologies mainly aimed at understanding and improving existing systems, they also proofed to be very useful for aligning various, successfully running systems during our experiments. By highlighting the individual configurations on the model, users can detect similarities and differences among the process variants as well as new possible configurations far easier than if they have to compare separate models.

In future research we have to setup experiments with larger real-world data to provide further guidance into the practical usage of the suggested methods as well as to test the applicability of further machine learning techniques. For example, we expect that the mining of association rules among different configurations can provide insights on interdependencies between configuration decisions and thus be used for further guidance on making configuration decisions. The re-play of log files generated from systems that are already based on a configurable reference model might help improving the configurable model over time.

Acknowledgements. We thank Antal van den Bosch and Ton Weijters for providing us with insights into the various machine learning techniques.

References

1. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In *Petri Nets and Other Models of Concurrency ICATPN 2007*, volume 4546 of *LNCS*, pages 484–494. Springer, 2007.
2. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Technische Universiteit Eindhoven, 2006.
3. J. Becker, P. Delfmann, and R. Knackstedt. Adaptive Reference Modelling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In *Reference Modeling*, pages 27–58. Springer, 2007.
4. P. Cabena, P.Hasjinian, R. Stadler, J. Verhees, and A. Zanasi. *Discovering data mining: from concept to implementation*. Prentice-Hall, Upper Saddle River, NJ, USA, 1998.
5. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Mining: Aggregating Instance Graphs into EPCs and Petri Nets. In *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 35–58. Florida International University, Miami, FL, USA, 2005.
6. B.F. van Dongen, M.H. Jansen-Vullers, H.M.W. Verbeek, and W.M.P. van der Aalst. Verification of the SAP Reference Models Using EPC Reduction, State-space Analysis, and Invariants. *Computers in Industry*, 58(6):578–601, 2007.
7. Eindhoven University of Technology. The ProM framework. http://prom.sf.net/.
8. P. Fettke and P. Loos. Classification of Reference Models – a Methodology and its Application. *Information Systems and e-Business Management*, 1(1):35–53, 2003.
9. F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Merging Event-driven Process Chains. BPM Center Report BPM-08-08, BPMcenter.org, 2008.
10. F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable Workflow Models. *International Journal of Cooperative Information Systems*, 17(2):177–221, June 2008.
11. G. Keller and T. Teufel. *SAP R/3 Process-oriented Implementation: Iterative Process Prototyping*. Addison Wesley Longman, Harlow, UK, 1998.
12. D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, San Francisco, CA, USA, 1999.
13. M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, March 2007.
14. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.
15. A. Rozinat, A.K. Alves de Medeiros, C.W. Günther, A. J. M. M. Weijters, and W.M.P. van der Aalst. The Need for a Process Mining Evaluation Framework in Research and Practice. In *Business Process Management Workshops*, volume 4928 of *LNCS*, pages 84–89. Springer, 2008.
16. O. Thomas, B. Hermes, and P. Loos. Towards a Reference Process Model for Event Management. In *Business Process Management Workshops*, volume 4928 of *LNCS*, pages 443–454, 2008.
17. S. Zhang, C. Zhang, and Q. Yang. Data Preparation for Data Mining. *Applied Artificial Intelligence*, 17(5):375–381, 2003.