Improving Business Process Quality through Exception Understanding, Prediction, and Prevention

Daniela Grigori

LORIA, INRIA Lorraine
Campus Scientifique, BP 239
Vandoeuvre les Nancy, 54506
France
Daniela.Grigori@loria.fr

Fabio Casati, Umesh Dayal, Ming-Chien Shan

HP Labs
1501 Page Mill Road
Palo Alto, CA, 94304
USA
[casati,dayal,shan]@hpl.hp.com

Abstract

Business process automation technologies are being increasingly used by many companies to improve the efficiency of both internal processes as well as of e-services offered to customers. In order to satisfy customers and employees, business processes need to be executed with a *high* and *predictable* quality. In particular, it is crucial for organizations to meet the Service Level Agreements (SLAs) stipulated with the customers and to foresee as early as possible the risk of missing SLAs, in order to set the right expectations and to allow for corrective actions.

In this paper we focus on a critical issue in business process quality: that of analyzing, predicting and preventing the occurrence of *exceptions,* i.e., of deviations from the desired or acceptable behavior. We characterize the problem and propose a solution, based on data warehousing and mining techniques. We then describe the architecture and implementation of a tool suite that enables exception analysis, prediction, and prevention. Finally, we show experimental results obtained by using the tool suite to analyze internal HP processes.

1. Introduction and motivations

Process design, automation, and management technologies are being increasingly used in both traditional and newly-formed, Interned-based enterprises in order to improve the quality and efficiency of their administrative and production processes, to manage e-commerce transactions, and to rapidly and reliably deliver services to businesses and individual customers.

In order to attract and retain customers as well as business partners, organizations need to provide their services (i.e., execute their processes) with a high, consistent, and predictable quality. From a process automation perspective, this has several implications: for example, the business processes should be correctly designed, their execution should be supported by a system that can meet the workload requirements, and the (human or automated) process resources should be able to perform their work items in a timely fashion. In this paper we focus on a specific, although critical issue in business process quality: that of reducing the occurrence of exceptions.

The term *exception* has been used with several different meanings in the workflow and process automation communities. We define an exception as a deviation from the "optimal" (or acceptable) process execution that prevents the delivery of services with the desired (or agreed) quality[1]. This is a high-level, user-oriented notion of exception, where it is up to the process designers and administrators to state what they consider as an exception, and hence as a problem they would like to address and avoid. Delays in completing an order fulfillment process or escalations of complaints to a manager in a customer care process are typical examples of exceptions. In the first case a company is not able to meet the service level agreements, while in the second case the service is delivered with acceptable quality from the customer's point of view, but with higher operating costs, and therefore with unacceptable quality from the service provider's perspective.

**Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001**

[1] We are concerned with both *external* quality (as perceived from the consumer in terms of better and faster services) and *internal* quality (as perceived by the service provider in terms of lower operating cost).

In this paper we present a tool suite that supports organizations in *analyzing*, *predicting*, and *preventing* exceptions. Exception analysis helps users in determining the causes of exceptions. For example, the analysis may show that delays in a supply chain process occur whenever a specific supplier is involved. Understanding the causes of exceptions can help IT and business manager to identify the changes required to avoid future occurrences of the exceptions. For example, the company may decide to remove a given supplier from its list.

The tool suite can also dynamically *predict* the occurrence of exceptions at process instantiation time, and progressively refine the prediction as process execution proceeds and more information become available. Exception prediction helps in setting the right expectations about the process execution quality. In addition, it allows users and applications to take actions in order to *prevent* the occurrence of the exceptions. For example, when the tool predicts that a process instance has a very high probability of missing its deadline, it can raise the process instance priority (depending on the importance of the process and on the potential damage caused by missing the deadline), in order to inform resources that work items of this process instance should be executed first.

Our approach is based on applying data mining and data warehousing techniques to process execution logs. In fact, business process automation systems (also called Workflow Management Systems, or simply WfMSs) record all important events that occur during process executions, including the start and completion time of each activity, its input and output data, the resource that executed it, and any failure that occurred during activity or process execution. By cleaning and aggregating workflow logs into a warehouse and by analyzing them with data mining technologies, we can extract knowledge about the circumstances in which an exception occurred in the past, and use this information to explain the causes of its occurrence as well as to predict future occurrences within running process instances.

This work is part of a larger research effort aiming at developing business intelligence techniques and tools for business process reporting, analysis, prediction, and optimization. Examples of other challenging problems we are addressing by using the same methodology include process definition discovery, intelligent assignment of activities to resources, and automatic identification of system requirements based on workload prediction. In the remainder of the paper we will refer to this research area as *Business Process Intelligence*, or BPI for short. Although very appealing from a business perspective, the development of a BPI solution presents many technical challenges:

- The main issues to be faced are the characterization of the problem, the identification of the technologies that can support our effort, and the composition of these technologies in an overall architecture.
- A prerequisite for process data analysis is the availability of a process data warehouse. The design, population, and maintenance of the warehouse are themselves quite complex problems, as discussed in [Casati01].
- Our goal is to design and develop the BPI tool suite as a ready-to-go solution. As such, it must be applicable in different conditions and must be capable of meeting a wide set of BPI requirements.

We decided to initially focus on issues related to exceptions, both because this is a very useful contribution in its own right, and because it is a complex problem, so that it can test our assumptions as well as the applicability of the overall approach. The specific problem of analyzing, predicting, and preventing exceptions presents additional challenges:

- The notion of what characterizes a "normal" versus an "exceptional" process execution varies depending on the business and IT needs. We need to be able to analyze and predict a broad range of situations in which users may be interested.
- We need to determine how exactly exceptions can be analyzed and predicted. In particular, we need to identify which data mining techniques can be applied, and which process attributes (features) should be provided as input to the data mining algorithms.
- The problem of exception prediction is particularly complex, since ideally we want to make the best possible predictions at every process execution stage. Therefore, we need to build predictive models tailored to the different process execution stages.

In this paper we detail these challenges, we discuss how to address them, and we describe techniques and tools for exception analysis, prediction, and prevention.

2. Related Work

To the best of our knowledge, there are no approaches to exception analysis and prediction based on data warehousing and data mining techniques, and in general there are very few contributions in the BPI area. Prior art exists in the field of exception prediction, limited however to estimating deadline expirations and based on simple statistical techniques. In the following we summarize these contributions, and then we underline the main differences with the approach proposed in this paper.

One of the first contributions to process time management is provided in [Panagos97], and builds on work in the real-time system domain. The authors address the problem of predicting as early as possible when a process instance is not likely to meet its deadline, in order to escalate the problem and take appropriate actions. In the proposed process model, every activity in the process has a maximum duration, assigned by the process designer based on the activity's estimated execution times and on the need to meet the overall process deadline.

When the maximum duration is exceeded, the process is escalated. When an activity executes faster than its maximum duration, a slack time becomes available that can be used to dynamically adjust the maximum durations of the subsequent activity. This activity can take all the available slack or a part of it, proportional to its estimated execution time or to the cost associated to escalating deadline expirations.

In [Eder99] the authors present another technique for deadline monitoring and management. In the proposed approach, a process definition includes the specification of the expected duration for each activity. This duration can be defined by the designer or determined based on past executions. In addition, the designer may define deadlines for activities or for the whole process. Deadlines specify the latest allowed completion times for activities and processes, defined as interval elapsed since the process instance start time. Processes are translated into a PERT diagram that shows, for each activity, based on the expected activity durations and on the defined deadlines, the earliest point in time when the activity can finish as well as the latest point in time when it must finish to satisfy the deadline constraints. During the execution of a process instance, given the current time instant, the expected duration of an activity, and the calculated latest end time, the progress of the process instance can be assessed with respect to its deadline. This information can be used to alert process administrators about the risk of missing deadlines and to inform users about the urgency of their activities.

Our approach differs considerably from the ones presented above. In fact, we aim at predicting any kind of exception, rather than focusing on deadline expirations. In addition, we propose to build prediction models by leveraging data warehousing and data mining techniques, that enable more accurate predictions, based on many characteristics of the process instances, such as data values, resources, the day of the week in which processes or activities are started, and many others.

In addition, the approach presented in this paper also allows exception analysis, to help users in understanding and removing the causes of the exception. A further difference with respect to the above-mentioned proposals is that this paper also presents the architecture and implementation of a tool suite for exception analysis and prediction, and illustrates experimental results. This approach and tool suite will be re-used and extended to provide more BPI functionalities.

Finally, we also mention the work by Hwang et al [Hwang99], since it also deals with exceptions and with the analysis of process execution log. However, the goal of the authors is to support users in *handling* exceptions once they have occurred. Exception handling suggestions are given by providing users with information about the way in which similar situations were handled in previous executions. Our goal is quite different, since we aim at analyzing exceptions to understand why they occur, and in predicting and preventing their occurrence, rather then in handling them.

3. Process Models and Process Execution Logs

There are hundreds of commercial WfMSs available on the market, as well as many research prototypes. While each system has a different process model and log structure, most of them share the same basic concepts. In this section we will present the process model and execution log structure of *HP Process Manager4.0* (HPPM), since this is the WfMS on top of which we built our prototype and conducted the experiments described in this paper. However, the same concepts and techniques are applicable to virtually any other WfMS.

In HPPM, a process is described by a directed graph, that has four different kinds of nodes:

- *Work nodes* represent the invocation of activities (also called *services*), assigned for execution to a human or automated *resource*.
- *Route nodes* are decision point that route the execution flow among nodes based on an associated *routing rule*.
- *Start nodes* denote the entry point to the processes.
- *Complete nodes* denote termination points.

Arcs in the graph denote execution dependencies among nodes: when a work node execution is completed, the output arc is "fired", and the node connected to this arc is activated. Arcs in output to route nodes are instead fired based on the evaluation of the routing rules.

As an example, Figure 1 shows the Expense Approval process, to which we will refer later in the paper when presenting experimental results. This is a simplified version of the actual process used within HP to request approval for various kinds of expenses. The process is started by the requester, who also specifies the expense amount, the reasons, and the names of the clerks and managers that should evaluate the request. Next, an email is sent to the requester to confirm the start of the process. The process then loops among the list of selected clerks and managers, until either all of them approve the expense or one of them rejects it. Finally, the result is notified to the requester.

Every work node is associated to a *service description*, that defines the logic for selecting a resource (or resource group) to be invoked for executing the work. The service also defines the process data items to be passed to the resource upon invocation and received from the resource upon completion of the work. Several work nodes can be associated to the same service description. For example, nodes *Notify Requester of Initiation*, *Notify Requester of Approval*, and *Notify Final Decision* of Figure 1 are all associated to service description *send_email*, executed by the resource *email_server*.

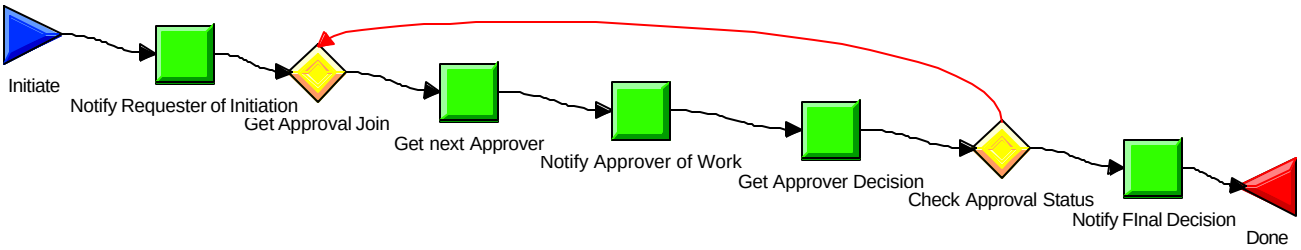

Figure 1 – The Expense Approval process

When a work node is scheduled for execution, the WfMS reads the corresponding service description, executes the resource selection rule associated to the service description, and puts the work item to be performed into the resource's *worklist*. Resources periodically connect to WfMS, pick a work item assigned to them (or to a group to which they are member of), and then execute it. Details on the HPPM process model are provided in [HPPM-PD]. An introduction to WfMSs in general is provided in [Leymann00].

WfMSs log information on process executions into an *audit log* database, typically stored in a relational DBMS. The audit log database include information on *process instances* (e.g., activation and completion timestamps, current execution state, name of the user that started the process instance), *service instances* (e.g., activation and completion timestamps, current execution state, name of the resource that executed the service, name of the node in the context of which the service was executed), and *data modifications* (e.g., the new value for each data item every time it is modified.)

A complete and detailed description of the HPPM audit log database schema is provided in [HPPM-TR].

4. BPI Architecture

This section presents the overall architecture of the BPI tool suite, to introduce the environment in which the work described in this paper has been developed. The BPI suite is composed of the *warehouse* of process definition and execution data, the *BPI engine*, and the *Monitoring and Optimization Manager*, or *MOM* (Figure 2).

Data are periodically extracted from the WfMS logs and loaded into the warehouse by Extract, Transfer, and Load (ETL) scripts. The warehouse is designed to support high-performance multidimensional analysis of process execution data possibly coming from heterogeneous sources. Hence, the warehouse of process execution data is a very useful component in itself, providing a wide range of reporting functionalities still missing in commercial WfMSs. A more detailed discussion of the BPI warehouse is provided in [Casati01].

The *BPI engine* executes data mining algorithms on the warehouse data in order to:

– Understand the causes of specific behaviors, such as the execution of certain paths in a process instance, the use of a resource, or the (in)ability to meet service level agreements.
– Generate prediction models, i.e., information that can be used to predict the behavior and performances of a process instance, of the resources, and of the WfMS.

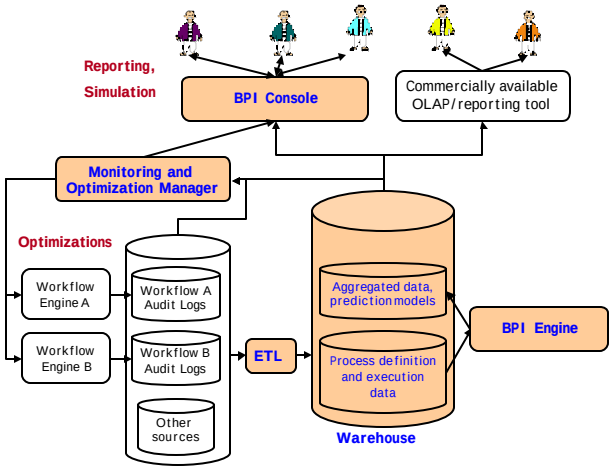

Figure 2 - Overall architecture of the BPI solution

The BPI engine stores the extracted information in the warehouse itself, so that it can be easily and efficiently accessed through the BPI console or through external OLAP and reporting tools.

The *MOM* accesses information in the warehouse as well as *live* information about running process instances stored in the WfMS logs to make predictions and dynamically optimize process instance executions. For example, MOM can be configured to raise the priority of a process instance when there is a high probability that the instance will not finish on time. MOM can also alert process administrators about foreseen critical situations.

5. Exception Analysis

This section describes our approach to supporting IT and business users in *understanding* the causes of exceptions. We begin the section by presenting our notion of "exception". Then, we provide an overview of our approach to exception analysis, before diving into the details and into the description of the implementation. Finally, we illustrate experimental results obtained by

applying the BPI tool suite to analyze exceptions in HP internal processes.

5.1 On the notion of Exception

Our approach is agnostic about what is or is not an exception. The user is free to state that a given event or situation is "exceptional", and analyze it. Exceptions are defined by conditions over process execution data. If an "exceptional" condition holds for a process instance, then the instance is labeled as affected by the exception. BPI provides a wide range of exception types (e.g., instances lasting more than D days, or being in the slowest X%, or in which node N was executed more than T times). Users can then configure the BPI to analyze a specific exception on a specific process. For instance, they can analyze instances of the *Expense Approval* process lasting more than *8* days (We will refer to this exception when presenting experimental results). BPI will then access process instance data, check the condition, and label the selected instances as being affected by the exception.

If users need to monitor exceptional behaviors not included in the built-in set, they can add new exception types by defining the "exceptional" condition. Details on the definition and implementation of exceptional behaviors and are provided in [Casati01].

5.2 Exception Analysis Overview

Exception analysis is performed by applying data mining techniques to process definition and execution data, collected in the warehouse. Specifically, we treat this problem as a *classification* problem.

Classification applications take as input a labeled training data set (typically in the form of a relational table) in which each row (tuple) describes an object (e.g., a *customer* in a customer management application) and the *class* to which this object belongs[2] (e.g., "profitable", "neutral", or "unprofitable" customer). The classifier then produces a set of *classification rules*, i.e., mappings from a condition on the objects' attributes to a class, with the meaning that objects whose attributes satisfy the condition belong to the specified class. Therefore, classification rules identify the characteristics of the objects in each class, in terms of values of the objects' attributes. For example, the classifier may discover the following classification rule: customers from Virginia and with a yearly salary over 50.000$ are "profitable". For each classification rule, the classifier also provides information about the rule's *accuracy*, i.e., about the probability that classifications performed with the rule are correct.

The exception analysis problem can be mapped to a classification problem, where process instances are the *objects*, that belong to either the "normal" or to the "exceptional" *class*. We are interested in finding classification rules that identify which are the characteristics of "exceptional" process instances. Once these characteristics have been identified, the user can have a much better understanding of the causes of the exception, and can then try to address such causes.

The approach to analyze why instances of a certain process are affected by a specific exception is composed of four phases.

The ***process data preparation*** phase selects the process instance attributes to be included as part of the input data set analyzed by the classifier. Relevant attributes can for example include the values of process data items at the different stages during process instance execution, the name of the resources that executed activities in the process instance, the duration of each activity, or the number of times a node was executed. Once the attributes of interest have been identified, then a data structure (typically a relational table) is created and populated with process instance execution data.

The ***exception analysis preparation*** phase joins in a single view the information generated by the previous phase with the exception labeling information (stating whether the instance is exceptional or not), computed by BPI at exception definition time.

The ***mining*** phase applies classification algorithms to the data generated by the data preparation phase.

Finally, in the ***interpretation*** phase, the analyst interprets the classification rules to understand the causes of the exception, and in particular to identify problems and inefficiencies that can be addressed and removed.

A few iterations of the mining and interpretation phases may be needed in order to identify the most interesting and effective classification rules. In particular, the mining phase may generate classification rules that classify process instances based on attributes that are not interesting in the specific case being considered. For example, the classification rules will certainly identify a correlation between the process instance duration and a deadline expiration exception. However, this is an obvious and not interesting correlation. Hence, the analyst may want to repeat the mining phase and remove the process instance duration attribute from the ones considered in generating the classification rules, so that the classifier can focus on "interesting" attributes.

In future versions of our applications we plan to make the process data preparation phase more "intelligent", so that it selects different attributes based on the kind of exception being analyzed (in the current version, the process data preparation phase is process–specific but exception-independent). However, we anticipate that this will be a difficult problem and that a few human-driven, attribute-purging iterations will always be needed.

5.3 Details and Implementation

This section details the proposed approach to exception analysis and presents our implementation, built on top of

[2] The object/class relationship described here should not be confused with that of object-oriented progaramming.

Oracle 8i (most of the BPI tool suite is written in SQL or PL/SQL). All the exception analysis modules are part of the *BPI engine* component of Figure 2.

The first phase (*process data preparation*) is particularly challenging. In fact, classification applications typically require input data to reside in a relational table, where each tuple describes a specific object. Therefore, to analyze why an exception affects instances of a process, we need to prepare a process-specific table (called *process analysis* in the following), that includes one row per process instance, and where the columns correspond to process instance attributes. One additional column is needed to store labeling information.

However, unlike traditional classification problems, the information about a single object (process instance) in the BPI warehouse is scattered across multiple tables, and each table may contain multiple rows related to the same instance. Hence, we are faced with the problem of defining a suitable process analysis table and of populating it by collecting process instance data.

In addition, even within the same process, different instances may have different attributes. The problem here is that a node can be activated a different number of times in different instances. The number of such activations is a-priori unknown. Hence, not only do we have to identify which are the interesting node execution attributes to be included in the *process analysis* table, but also how many node executions (and which ones) should be represented.

This issue can be addressed in several ways: for example, we could decide that if a node can be activated multiple times, then we consider for our analysis only a specific node execution (e.g., the first one or the last one). An alternative approach consists in considering all node executions. In this case, the process analysis table must have, for each node, a number of columns proportional to the maximum number of executions of that node, determined by looking at the process instance data in the warehouse. However, despite the fact that this technique provides more information to the mining phase, it does not necessarily give better results. In fact, tables generated in this way typically includes many undefined (NULL) values, especially if the number of node activations greatly differs from instance to instance. Commercial data mining tools do not suitably manage sparse tables. In addition, when classifications are based on a large number of similar attributes that often have null values, it is very difficult to interpret and understand the results. Finally, this approach can computationally heavy.

The approach we followed consists in inserting two attribute (column) sets for each node that can be executed multiple times: one to represent the *first* execution and the second to represent the *last* execution of that node. This is due to the observation, from several experiments we have conducted on different processes, that the first and last executions of a node in the process have a higher correlation with many kind of process exceptions, such as those related to process execution time and to the execution of a given subgraph in the process.

Finally, we observe that the number of process instance attributes of interest for our purposes is in general unlimited. For example, an exception could be related to the ratio between the durations of two nodes in the process, or to the sum of two numeric data items. In our implementation we have configured the tool to select the attributes that have shown higher correlations with exceptions in the tests we have performed. In particular, the *process analysis* table includes the following attributes for each process instance:

- *Activation and completion timestamps:* these actually corresponds to multiple columns, that decompose the timestamps in hour of the day, day of the week, etc., and with the addition of the *holiday* flag to denote whether the process was instantiated on a holiday.
- *Data items:* Initial values of the process data items, plus the length (in bytes) of each item.
- *Initiator:* Resource that started the process instance.
- *Process instance duration.*

In addition, the process analysis table includes attributes for each node in the process (two sets of attributes are included for nodes that can be executed multiple times, as discussed above):

- *Activation and completion timestamps* (decomposed as described for the process instance timestamps).
- *Data items:* Values of the node output data, plus the length (in bytes) of each item.
- *Resource that executed the node.*
- *Final state of the node* (e.g., completed or failed)
- *Node duration.*
- *Number of activations* of the node in the process instance (this attribute is only included once per node, even if two attribute sets are used for this node, since the value would be the same for both).

The process analysis table is automatically built by a *process analysis preparation* PL/SQL script. This script takes the name of the process to be analyzed as input parameter, and retrieves process definition information from the BPI warehouse. In particular, the script identifies the nodes and data items that are part of the process, and creates the *process analysis* table. Then, the script populates the table with process instance data. Users can also restrict the process analysis table to contain only data about instances started within a time interval.

The *exception analysis preparation* phase is implemented by process- and exception-independent PL/SQL code that receives as parameter the name of the process and of the exception to be analyzed, and generates a process- and exception-specific view. The view joins the *Process Analysis* and *ProcessBehaviors* tables (the latter is a process- and exception-independent table that lists which instances have been affected by which exceptional

behaviors), to provide a data set that includes process instance attributes as well as labeling information. The obtained view includes all the information required by the classification tool to generate the classification rules.

The *mining* phase can be performed by using different algorithms and techniques. A variety of data mining and classification applications are available on the market. Therefore we did not develop our own mining algorithms, but expect that a commercial tool be employed, at least in the preliminary versions of the BPI tool suite.

In particular, we typically use *decision trees* [Berry00] for exception analysis. Decision trees are widely used because they work well with very large data sets, with large number of variables, and with mixed-type data (e.g., continuous and discrete). In addition, they are relatively easy to understand (even by non-expert users), and therefore simplify the *interpretation* phase. With decision trees, objects are classified by traversing the tree, starting from the root and evaluating branch conditions (decisions) based on the value of the objects' attributes, until a leaf node is reached. All decisions represent partitions of the attribute/value space, so that one and only one leaf node is reached. Each leaf in a decision tree identifies a class. Therefore, a path from the root to a leaf identifies a set of conditions and a corresponding class, i.e., it identifies a classification rule. Leaf nodes also contain an indication of the rule's accuracy, i.e., of the probability that objects with the identified characteristics actually belong to that class. Decision tree building algorithms in particular aim at identifying leaf nodes in such a way that the associated classification rules are as accurate as possible.

Once a decision tree has been generated by the mining tool, analysts can focus on the leaf nodes that classify instances as exceptional. Then, they can traverse the tree from the root to the leaf, to identify which attributes and attribute values lead to the leaf node, and therefore identify the characteristics of "exceptional" instances.

5.4 Experimental Results

We have applied the BPI approach and toolkit to analyze several administrative processes within HP, such as electronic employee reimbursements and requests for disbursement vouchers. These processes are implemented on top of HP Process Manager, and are accessed by hundreds of employees per day. As a representative example, we discuss the results obtained in analyzing the *Expense Approval* process described in Figure 1, and specifically in identifying the characteristics of instances that take more than 8 days to complete (the average execution time was about 5 days). We had access to five months of process execution data, corresponding to approximately 50.000 process instances. About 15% of the instances were affected by this "exception".

After importing the process instance data into the BPI warehouse and having defined the exception, we ran the scripts described in the previous section to label the instances and generate the exception analysis table. We next used SAS Enterprise Miner (a leading commercial data mining tool) for the generation of decision trees. In the preparation of the decision tree we used $?^2$ as splitting criteria, and we used the proportion of correctly classified records as assessment value. 60% of records were used as training data while 40% of records were used for validation. These are the parameters that gave us the best overall results.

After several failures, that led us to restructure database schemas and preparation scripts as described earlier in this section, the tool finally produced interesting results. For both simplicity and confidentiality reasons, we do not show the exact decision tree and some details of the results. However, we summarize the main findings in Figure 3. The results show the following:

- Cases that required many expense approvers were more likely to last longer. In particular, exceptional instances typically had more than 6 approvers.
- When, in addition to having more than 6 approvers, clerical activities[3] were executed by employees in a specific group, then 70% of the instances were "exceptional". The majority of process instances were instead on time when such clerical activities were executed by other employees.
- Process instances started on Fridays were more likely to last longer, since the work was in fact postponed until the next Monday.

As it often happened in our analysis, some of the identified correlations are not immediately helpful in order to understand and resolve the problem. For example, the fact that processes with more approvers (and therefore more node executions) last longer is to be expected. If all the identified correlations are of this kind, then it is very likely that what we classified as "exception" instead it is not an exception, but simply something that is part of the nature of the process.

However, some of the identified correlations are often useful to isolate bottlenecks and, in general, aspects of the process or of the organization that can be improved. For example, following the discovered correlation between exceptions and the resource group, a further look at the BPI warehouse revealed that employees in that group had more workload than others. Hence, the analysis allowed spotting a problem and suggesting a possible solution, by reassigning the work appropriately.

Business process intelligence, just like any other business intelligence application, requires care in interpreting the results and in identifying biases due to the kind of data that is available. For example, a problem characteristic of BPI is the *border* effect: typically, the analysis is performed on processes started (or completed, or both) within a certain time window. For example, we may have a data set containing all instances completed in

[3] The first approver is typically a clerck that verifies that the request is formally correct and that payments can be made.

October. If a mining tool analyzes these data, it will determine that instances started in spring lasted longer than those started in the summer or fall. Indeed, the tool will state that the accuracy of this rule is very high. However, the result is only due to how data are collected, rather than to a property of the process: in fact, the data set is polarized with respect to the start date, in that it contains instances started in spring only if they lasted very long, i.e., until October. A formal analysis of this and other typical biases is also part of our future research agenda.

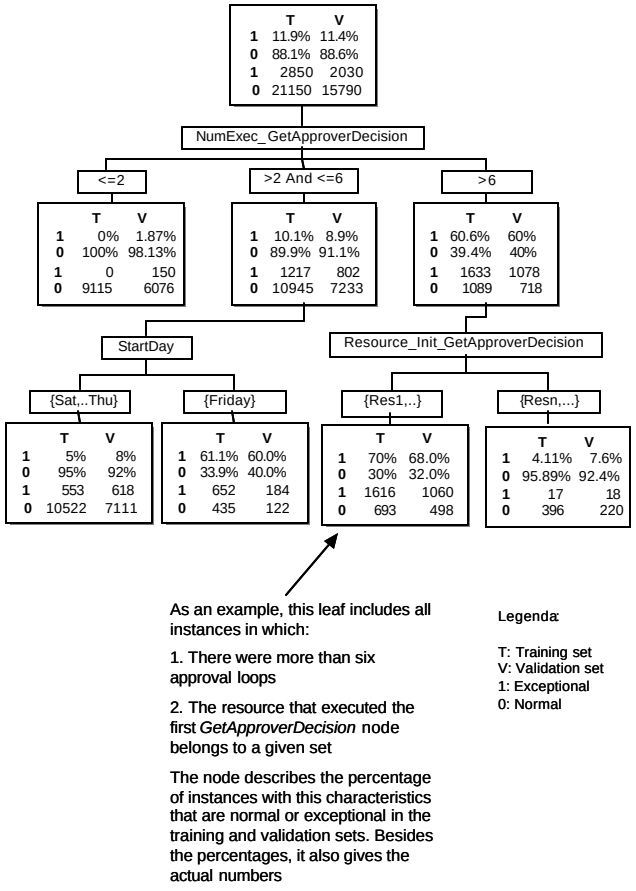

Figure 3 - Simplified decision tree obtained by analyzing the Expense Approval process.

6. Predicting and Preventing Exceptions

One of the goals of our work is that of *predicting* the occurrence of exceptions. In particular, we aim at predicting exceptions as early as possible in process executions, so that they can be *prevented*, or so that at least adequate expectations about the process execution speed and quality can be set. As in the previous section, we first provide an overview of the approach. Then, we detail the approach and describe our implementation. Finally, we illustrate experimental results.

6.1 Overview

The problem of exception prediction has many similarities with that of exception analysis. In fact, exceptions could be predicted by identifying the characteristics of exceptional instances, and by then checking whether a running process instance has those characteristics. Indeed, our approach to exception prediction includes the four phases described in the previous section. However, there are a few differences that must be taken into account. In particular, the *process data preparation* phase must face additional challenges in the context of exception prediction.

The problem is that classification rules generated by exception analysis work very poorly (and may not even be applicable) for predictions about running instances. In fact, we want to classify process instances as "normal" or "exceptional" while they are in progress, and possibly in their very early stages. Therefore, the value of some attributes (such as the executing resource or the duration for a node yet to be executed) may still be undefined. If the classification rules generated by the exception analysis phase include such attributes, then the rules cannot be applied, and the process instance cannot be classified. For example, assume that decision tree-building algorithms have been used in the mining phase. If undefined attributes appear in the branch conditions of the decision tree, then the branch condition cannot be evaluated. The prediction becomes less accurate as the undefined attributes appear in branch conditions closer to the root of the tree, since we can only follow the tree (and improve the classification accuracy) while branch conditions can be evaluated. At an extreme, if undefined attributes are in the branch condition at the root of the tree, then the decision tree does not give any useful information.

We address this issue by modifying the *process data preparation* phase so that it generates several different process analysis tables (that will eventually result in several different classification rule sets), each tailored to make predictions at a specific *stage* of the process instance execution. A stage is characterized by the set of nodes executed at least once in the instance. For example, the process analysis table targeted at deriving classification rules applicable at process instantiation time is prepared by assuming knowledge of only the process instance input data, the starting date, and the name of the resource that started the instance. In this way, only these attributes will appear in the classification rules.

The other phases are executed as discussed in the previous section, with the difference that they are performed once for every table generated by the process data preparation phase. In addition to the phases common with exception analysis, exception prediction also includes a *prediction* and a *reaction* phase.

The ***prediction*** phase is where predictions on running process instances are actually made. In this phase, classification rules are applied to live instance execution

data, to classify the instances and obtain, for each running instance and each exception of interest, the probability that the instance will be affected by the exception.

In the ***reaction*** phase, users or systems are alerted about the risk of the exception, and take the appropriate actions to reduce the "damage" caused by the exception, or possibly to prevent its occurrence.

6.2 Details and implementation

This section details the process data preparation, prediction, and reaction phases. The other phases are not discussed since they are performed and implemented as described in the previous section.

The *process data preparation* phase first determines the possible process instance stages, i.e., the different possible combinations of node execution states (*executed* or *not executed*). Then, for each stage, the process analysis table is constructed as described in the section 5. The first stage is always the one where no node has been executed, and is used to make predictions at process instantiation time. For this stage, the *process analysis* table will only contain information about the instantiation timestamp, the initial value of process data items, and the resource that started the instance. The process analysis tables generated for the other stages will include, for each executed node, the same node attributes listed in the exception analysis section. In the current implementation, for simplicity, we only consider the first execution of the node, so that at most one attribute set for each node is included. This phase is implemented through a PL/SQL script that takes the process name as input parameter and generates all the *process analysis* tables for that process.

The *prediction* phase is executed by the *Exception Monitor* (EM). The EM is part of the MOM component of Figure 2, and accesses both the BPI warehouse and the WfMS logs in order to make predictions. Access to WfMS logs is required since the BPI warehouse does not contain live data, but is instead updated periodically (typically once a day or once a month), depending on the business needs. Hence, while classification rules can be obtained "off-line", by analyzing warehouse data, the actual predictions need to be made on the live data that the WfMS writes in its logs. Access to the BPI warehouse is instead needed to retrieve the classification rules, generated beforehand. Indeed, our approach assumes that the mining phase stores its output in the database, so that rules can not only be interpreted by humans, but also used by applications such as the EM.

The EM operates by periodically accessing the WfMS audit logs and copying the tables containing information about process instance executions. This operation is quite simple and is executed on top of a relatively small database (since data are periodically purged from the audit log and archived in the warehouse). Hence, it has a negligible effect on the performance of the operational system. Once the data has been copied, the EM examines instances of processes to be monitored.

In particular, for each instance, the EM first determines the execution stage, by checking which nodes have been executed. Next, it accesses the BPI warehouse to retrieve the classification rules (that in our case have the form of a decision tree) to be applied, based on the execution stage. Once the appropriate decision tree has been identified, the EM scans the tree and evaluates each branch condition based on the value of the process instance attributes, until it reaches a leaf node. The leaf node will contain an indication of the probability that the examined instance is exceptional. If this probability is above a threshold, then a new tuple is inserted into a *warning* table, detailing the process instance identifier, the exception identifier, the execution stage, and the probability of the exception occurrence.

The *reaction* phase is executed by the *Exception Prevention Manager* (EPM), also part of the MOM. The EPM monitors the *warning* table. When a new exception is predicted for a process instance, the EPM alerts the user registered as the contact person for the process. Users can then perform actions on the WfMS or in the organization to try to prevent the exception or to reduce its impact. In addition, the EPM can be configured to proactively interact with the WfMS in an attempt to prevent the exception. Currently, the only allowed form of automated intervention consists in raising the process instance priority for those instances that are likely to be late. The process administrator can specify the level to which the priority can be raised depending on the probability of the process instance being late. In the future we plan to extend the EPM automatic reaction capabilities to:

- Modify process instance and work node priorities based on the risk and cost of missing SLAs.
- Modify the resource assignment policies so that activities are given to faster resources.
- Influence decision points in the process, so that the flow is routed on certain subgraphs, if this can help avoid the exception while still satisfying the customers and process goals (although perhaps causing in increased process execution costs).

6.3 Experimental Results

We now show initial results obtained by applying this approach to HP administrative processes. We will again refer to the Expense Approval process of Figure 1, and specifically to the same process duration exception described above. In the Expense Approval process, it was possible to have a good prediction for the instance duration at the very start of the instances. In fact, the resulting decision tree revealed that the length of the process is correlated to the name of the requester (i.e., the creator of the instance) and to the length of data item *Approver*, that contained the names of the approvers (and therefore its length indicated the number of loops to be

executed). For some combinations of these values, over 70% of the instances were affected by the exception[4].

As expected, predictions get more and more accurate as process instance execution proceeds, since more information about the instance becomes available. In particular, very accurate predictions could be made right after the execution of node *Get Approver Decision*. In fact, this activity is the most time-consuming one, and therefore after its execution it is possible to have more information about the likelihood of the process exceeding the acceptable execution time. The decision tree for predicting the duration exception at this stage of the process (depicted in Figure 4) shows in fact that if the first execution of node *Get Approver Decision* takes slightly more than 5 and half days, then the instance can be predicted as exceptional with 93% probability.

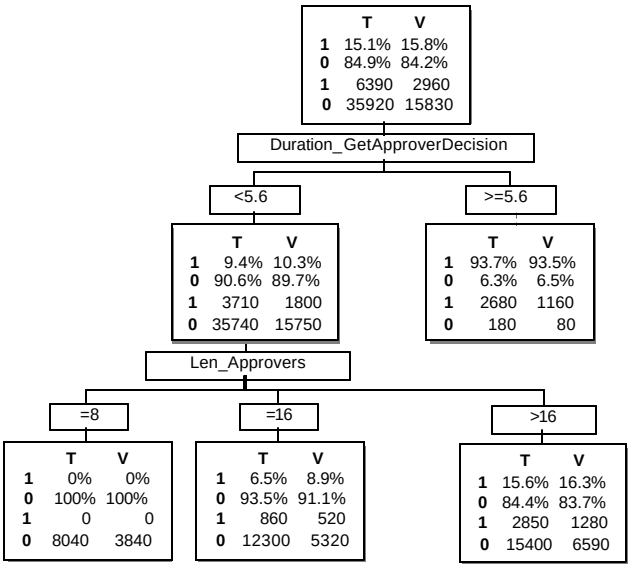

Figure 4 - Decision tree for exception prediction after the execution of node *Get Approver Decision*

We observe here that the duration of this node also appeared in the rules generated while *analyzing* this exception. However, in that context, we had removed this attribute from the input data set, since it was not particularly helpful in identifying "interesting" correlations and in understanding *why* the exception occurs. In this case instead, our focus is on predictions, and any correlation becomes useful.

In general, we experienced that there are only a few stages in a process instance where the accuracy of the prediction improves, typically after the execution of some "critical" nodes. An interesting optimization of our algorithms could therefore be based on the identification of such stages, so that the various exception prediction phases can be executed for these stages only.

[4] Given that only 15% of the instances is exceptional, predicting that exceptions with a 70% accuracy is a quite interesting result.

7. Concluding Remarks

This paper has presented an approach and a tool suite for exception analysis, prediction, and prevention. We have discussed the main challenges we had to face in undertaking this effort, and we have described how we have addressed them in our approach and implementation.

The experimental results have been (eventually) quite encouraging. Therefore, we plan to put a considerable effort in this research area and address the issues that still lie ahead. In particular, our research agenda includes the refinement of the exception prediction algorithms, to better handle the problem of multiple executions of a node within the same process instance. Eventually, as we gain more knowledge about the problem, we aim at developing new *classification* algorithms that are able to scan related data scattered among multiple tuples in multiple tables and extract classification rules, without the need of aggregating data in a single table. Other research objectives include a refined process data preparation phase that selects attributes also based on the exception being analyzed, the development of a complete methodology for exception analysis, and improved mechanisms for automated exception prevention.

The work presented in this paper is part of a larger, long-term research effort aiming at developing a Business Process Intelligence solution for WfMSs. Other research objectives in the BPI context include process definition discovery, execution path analysis and prediction, and dynamic system, process, and resource optimization.

8. References

[Berry00] M. Berry, G. Linoff. Mastering Data Mining. Wiley, 2000.

[Casati01] A. Bonifati, F. Casati, U. Dayal, M.C. Shan. Warehousing Workflow Data: Challenges and Opportunities. *Procs. of VLDB'01*. Rome, Italy. Sept. 2001.

[Eder99] J. Eder, E. Panagos, H. Pozewaunig, M. Rabinovich: Time Management in Workflow Systems. *Proceedings of BIS'99*. Poznan, Poland, 1999.

[HPPM-PD] Hewlett-Packard. HP Changengine Process Design Guide. Edition 4.4. 2000

[HPPM-TR] Hewlett-Packard. HP Changengine Technical Reference Guide. Edition 4.4. 2000

[Hwang99] S. Hwang, S. Ho, J. Tang. Mining Exception Instances to Facilitate Workflow Exception Handling. *Proceedings of DASFAA'99*, Taiwan, Apr. 1999.

[Leymann00] F. Leymann, D. Roller: *Production Workflow*. Prentice-Hall, 2000.

[Panagos97] E. Panagos, M. Rabinovich: Escalations in Workflow Management Systems. *Procs. of DART'97*, Rockville, Maryland, Nov. 1997.