2010

Performance Analysis of Business Process Models with Advanced Constructs

David Piessens
Eindhoven University of Technology, davidpiessens@gmail.com

Queensland University of Technology

Moe Thandar Wynn
Queensland University of Technology, m.wynn@qut.edu.au

Michael Adams
Queensland University of Technology, mj.adams@qut.edu.au

Boudewijn F. van Dongen
Eindhoven University of Technology, B.F.v.Dongen@tue.nl

Performance Analysis of Business Process Models with Advanced Constructs

David Piessens
Eindhoven University of Technology, Eindhoven, The Netherlands
Queensland University of Technology, Brisbane, Australia
Email: d.a.m.piessens@student.tue.nl

Moe Thandar Wynn
Michael Adams
Queensland University of Technology, Brisbane, Australia
Email: m.wynn@qut.edu.au; mj.adams@qut.edu.au

Boudewijn F. van Dongen
Eindhoven University of Technology, Eindhoven, The Netherlands
Email: B.F.v.Dongen@tue.nl

Abstract

The importance of actively managing and analysing business processes is acknowledged more than ever in organisations nowadays. Business processes form an essential part of an organisation and their application areas are manifold. Most organisations keep records of various activities that have been carried out for auditing purposes, but they are rarely used for analysis purposes. This paper describes the design and implementation of a process analysis tool that replays, analyses and visualises a variety of performance metrics using a process definition and its corresponding execution logs. The replayer uses a YAWL process model example to demonstrate its capacity to support advanced language constructs.

Keywords

Business Process Management, Performance Analysis, Process Mining, Yet Another Workflow Language (YAWL), Log Replayer.

INTRODUCTION

Business processes form the heart of every organisation, whether small or large, and a number of processes can always be identified within an organisation and their information systems. Organisations regularly undertake process improvement activities to ensure that their operational processes are as effective and efficient as possible (Weske 2007). Most organisations develop AS-IS and TO-BE process models of their operations to identify process improvement opportunities. Carrying out performance analysis on existing and planned process models offers a great way for organisations to detect flaws, bottlenecks, and other issues within their processes and allows them to make more effective decisions. For instance, Brataas, Hughes and Sølvberg (1997) presented a framework to measure the performance of workflows that involve both manual and automated activities.

Business process simulation is commonly used to carry out performance analyses of processes (Ardhaldjian and Fahner 1994). One of the drawbacks to using process simulation techniques is that the performance analysis results are only as good as the input data that is being used to generate the simulation experiments (i.e. garbage-in-garbage-out). Setting up realistic simulation experiments from scratch is very time-consuming. Furthermore, many business process simulation tools abstract from complex concepts such as *cancellation and multiple concurrent activity instances*, which occur in many business operations.

An executable process model, i.e., a workflow, contains descriptions of what tasks need to be performed, when they need to be performed, by whom they need to be performed, and what information they need and what information they produce. The work on workflow patterns by van der Aalst et al (2003a) illustrated the need for complex workflow constructs such as cancellation, multiple concurrent instances, and advanced synchronisation. Cancellation refers to the ability to stop or cancel current ongoing activities in certain parts of a process when an event occurs (e.g., a cancel order request made by the user). Multiple concurrent instances allow the same activity to be executed in parallel with different information (e.g., getting a price quote from different suppliers). Advanced synchronisation constructs allow merging of active concurrent threads of activities so that a workflow does not deadlock (Wynn 2009). These advanced features are present in industrial process models in different domains; a number of such processes are described in ter Hofstede et al. (2010).

Many process-aware information systems nowadays provide some kind of event log (also referred to as transaction log or audit trail) (van der Aalst et al. 2003b), where an event often refers to an activity (within a process instance); each event is logged with descriptors such as timestamp, event-type and executing resource. In the presence of such logs, a better alternative to process simulation is to make use of such log data for performance analysis. The resulting performance metrics can then be used to get a better understanding of business process performance, leading to critical review and potential redesign of processes as required.

This paper demonstrates how the performance analysis of complex process models can be carried out with the use of an *event log replayer*, an application that will replay all of the traces from an event log to calculate and store metrics related to performance analysis. The paper focuses on the design and the development of an event log replayer for processes modelled using the Yet Another Workflow Language (YAWL) notation (van der Aalst and ter Hofstede 2005), and the visualisation of the calculated performance metrics.

The remainder of the paper is organised as follows: first, we describe some background to the approach taken together with a purchase order process modelled in YAWL for illustrative purposes. Next, we describe the design framework of the YAWL process log replayer, followed by a detailed discussion of the prototype implementation. We also illustrate the performance analysis results based on the purchase order example. In conclusion, related work is discussed, improvements proposed and future work presented.

BACKGROUND

Process mining is a technology that uses event logs (i.e. recorded actual behaviours) to analyse executable business processes or workflows (van der Aalst et al. 2003; 2007) by referencing the process definition and the process-related information found in those logs. These techniques provide valuable insight into control flow dependencies, data usage, resource utilisation and various performance related statistics. This is a valuable outcome in its own right, since such dynamically captured information can alert us to problems with the process definition, such as 'hotspots' or bottlenecks that cannot be identified by mere inspection of the static model alone.

A much better insight of AS-IS process models can be achieved when we use those models for comparative analysis with the resulting event logs (i.e. log replay). Event log replayers are tools developed to replay, or even simulate an event log within a process. Ideally, this event log is retrieved directly from the tool in which the process was executed, but it could also be that the event logs are retrieved from an external source or other tool. While replaying, the replayer checks if traces are indeed executable in the process and analyses the information that is associated with the events contained in the trace. This analysis leads to the calculation of several performance metrics, which can be displayed to the process owner. These include information about the location of bottlenecks, which tasks are executed many times and are thus important, which cases never complete, and so on (Adriansyah 2009). One of the leading tools for process mining is the ProM[1] framework (Weijters et al. 2007) that offers a range of log replayers, each designed for a specific type of process model, such as Petri Nets or Fuzzy models (Adriansyah et al. 2010). However, none of the existing log replayers can handle processes that contain complex control flow dependencies such as cancellation, multiple concurrent instances, and advanced synchronisation and are thus insufficient to carry out performance analysis of process models with such features. In this paper, we demonstrate how log replay techniques can support such advanced concepts using processes modelled in the YAWL language. The *YAWL* language was created as the reference process modelling language for the well-known workflow patterns (van der Aalst et al. 2003a), whereby direct support is provided for complex control flow constructs such as multiple instances, cancellation regions and OR-joins (van der Aalst and ter Hofstede 2005). The YAWL workflow environment[2] is an open-source, state-of-the-art process automation system that supports the design, execution, and analysis of YAWL process models (van der Aalst et al. 2004; ter Hofstede et al. 2010). YAWL is currently being used in a wide variety of academic and industrial settings. The YAWL environment supports executable business processes that make use of several advanced constructs such as cancellation and multiple instances that are not easily supported in languages such as Business Process Modelling Notation (BPMN) and Event-Driven Process Chains (EPCs). The system provides support for execution of complex control flows with sophisticated resource allocation mechanisms. As such, YAWL provides an ideal environment to model complex processes and to generate detailed event logs during the execution.

To demonstrate how business processes can be modelled in YAWL, we present a fictitious purchase order process modelled as a YAWL workflow (see Figure 1). It features several typical control flow structures (such as XOR and AND splits and joins, iterations, and nested processes), as well as some of the complex constructs YAWL has to offer, such as multiple instances and cancellation regions.

[1] http://www.processmining.com
[2] http://www.yawlfoundation.org

A YAWL process has a unique starting point (input condition) and a unique end point (output condition). Tasks (activities) in the model are represented by rectangles (e.g. the *Receive purchase requisition* task), with all tasks on a directed path between the input and output conditions. A task may be an atomic task, representing a unit of work, or a composite task that is unfolded into another (sub-) process (e.g. the *Get approval* task). The split and join behaviours of a task are modelled by *decorators* attached to a task. A (red) dot inside the upper-right hand corner of a task indicates that it has a cancellation region defined for it (see below).

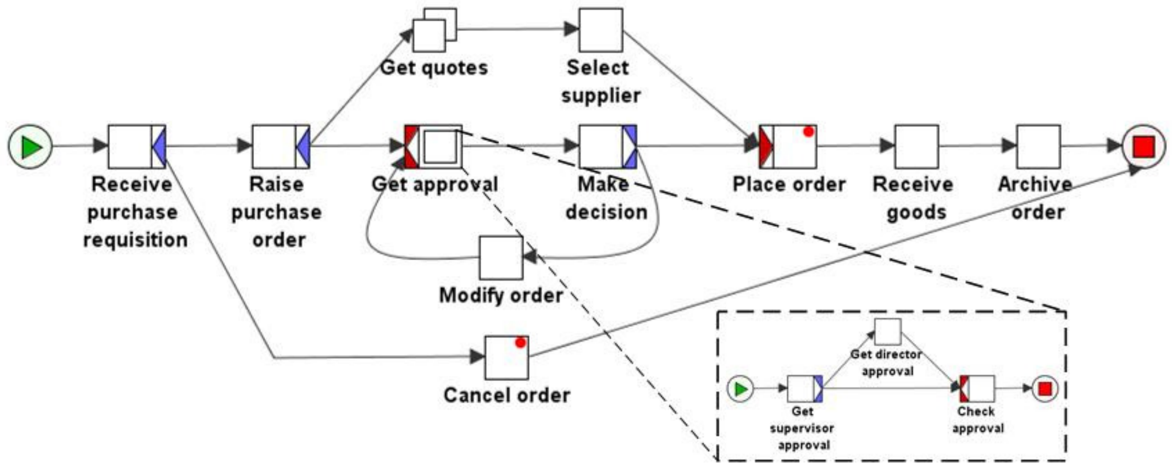


Figure 1: YAWL Diagram Purchase Order Process

The example process starts when a purchase requisition is received. After this activity is completed, the *Raise purchase order* task and *Cancel order* task are enabled in parallel (modelled as an AND-split decorator on the *Receive purchase requisition* task). Once enabled, the *Cancel order* task may be performed at any time to cancel an active purchase order, perhaps due to a user request or because the order does not receive approval. After raising a purchase order within the purchasing system, the order is assigned a unique order number. After completing the *Raise purchase order* task, two tasks can be done concurrently: getting quotes from suppliers and obtaining approval from management. Quotes are obtained from three different suppliers and one of these suppliers is then chosen to fill the order. If approval is not granted, the order can be modified and resubmitted for approval. When both tasks have been satisfactorily completed, the order is placed. After receiving the goods, the order is archived.

Complex constructs such as *Composite Task*, *Multiple Instances* and *Cancellation Regions* are present in the model. A composite task is a container for another YAWL (sub) Net, with its own set of YAWL elements constrained by the same syntax. The composite task in the purchase order process, *Get approval* (expanded in Figure 1), entails the request of an approval to sign off on a purchase order. In some cases, a supervisor's approval is enough, in other cases the director's approval is also required, depending on the details of a particular order.

The *Get quotes* task is an example of a Multiple Instance Atomic Task, which allows multiple instances of a task to be created and executed concurrently. In the *Get quotes* task, several suppliers are requested to supply a quote for the ordered goods. The task is configured to create a maximum of three supplier requests for quotations concurrently, and that when at least two quotations are received the process will continue (known as the *threshold*).

Cancellation regions are one of the unique features of the YAWL language. Any task can have a cancellation set associated with it. These cancellation sets may contain any number of other tasks and/or conditions in the same net that are nominated for cancellation upon the completion of the containing task. In this example, we like to model the fact that if the *Cancel order* task is completed, then any other tasks being performed at the same time (between *Raise purchase order* and *Place order* inclusive) should cease. This is shown in Figure 2, where task *Cancel Order's* cancellation region is depicted by dashed lines around the set of tasks it will cancel (*Raise purchase order*, *Get quotes*, *Select supplier*, *Get approval*, *Make decision*, *Modify order* and *Place order*). Similarly, to model the business rule that it should not be possible to cancel an order after the order has been placed, there is a cancellation region associated with the *Place order* task, which contains the *Cancel order* task.

YAWL PROCESS LOG REPLAYER FRAMEWORK

The YAWL process log replayer has been developed in *ProM* (Verbeek 2010), a generic open-source framework for implementing process mining tools in a standard environment. ProM 6 provides for the implementation of process analysis tools that can be "plugged-in" to the environment in a standard and structured way. An overview of the developed architecture and interoperability between the YAWL version 2.1 environment and ProM version 6 components can be found in Figure 3.

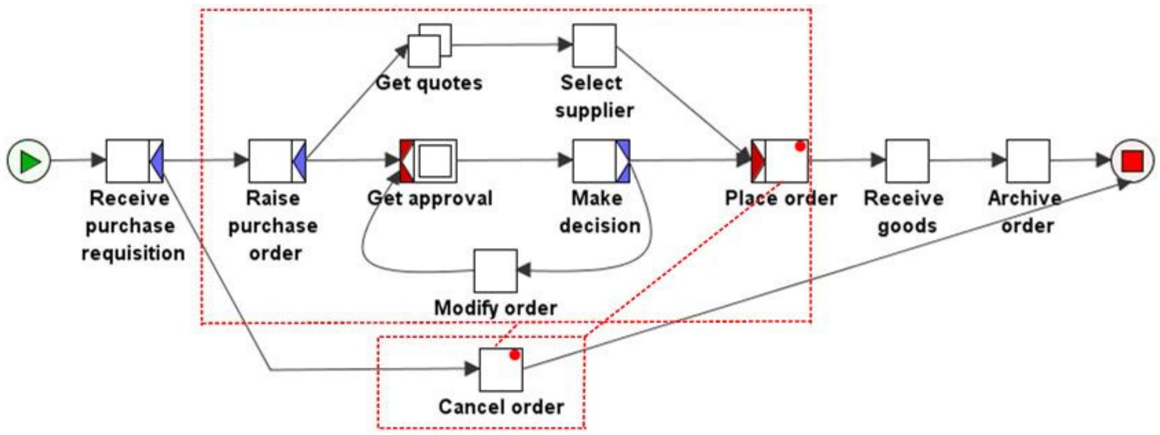


Figure 2: YAWL Purchase Order Process from Figure 1 with cancellation regions for tasks shown

Architecture

The work presented in this paper is implemented in the "YAWLReplay" package, which is available from the ProM repository[3] and can be installed using the package manager. The replay plug-in requires two input objects: a YAWL specification and its corresponding log file, as shown in Figure 4.

Executable YAWL models, including the control-flow, data and resource perspectives (Jablonski and Bussler 1996), can be created using the editor component of the YAWL environment. A YAWL process specification is uniquely identified by the combination of a system-generated key and a version number. The specification can then be uploaded and executed within the engine component. For each execution of a specification, data describing all aspects of its operation (events, transitions, allocated resources, data modifications, etc.) are recorded in process logs.

We made use of existing source code from the YAWL Editor to develop the specification import functionality for the replayer plug-in. The imported YAWL specification can then be viewed within the ProM environment using the open source graphical depiction library, JGraph[4].

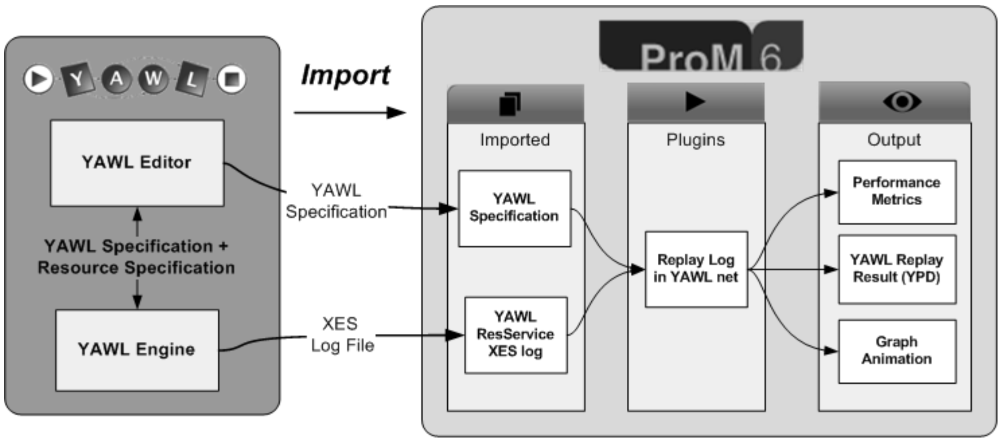


Figure 3: Architecture

The richest form of event logs supported by ProM 6 are those recorded in the XES format. OpenXES is a reference implementation of the XES XML-based standard for storing and managing event log data (Günther 2010). Since XES allows for the standardization of attributes through extensions, our approach can easily be extended to calculate metrics on other attributes than the ones we used. However, for this paper, we focussed on the data that is typically provided by YAWL.

[3] http://prom.win.tue.nl/ProM/packages/packages.xml
[4] http://www.jgraph.org

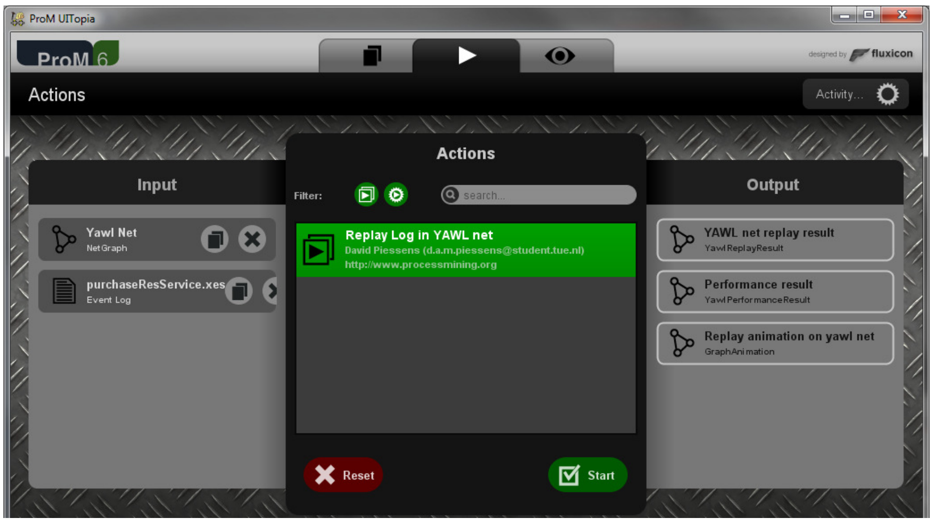


Figure 4: Starting the Replay plugin on a YAWL Net and a XES event log

Since YAWL stores its process logs in a number of relational database tables, new methods were added to the logging interface of the YAWL framework (version 2.1) that provided for the translation and the export of log data in the required XES format. In addition, the delegated service model of the YAWL environment means that, while the YAWL Engine manages (and logs) the control-flow and data perspectives, the resource perspective is managed separately by the discrete YAWL Resource Service, which produces its own process logs. Therefore, new methods were also added to the Resource Service's logging interface to enable XES formatted log export. Finally, a merged log functionality was developed which combines related log information from both the Engine and Resource Service into a single log export that fully describes all aspects of a specification's executions.

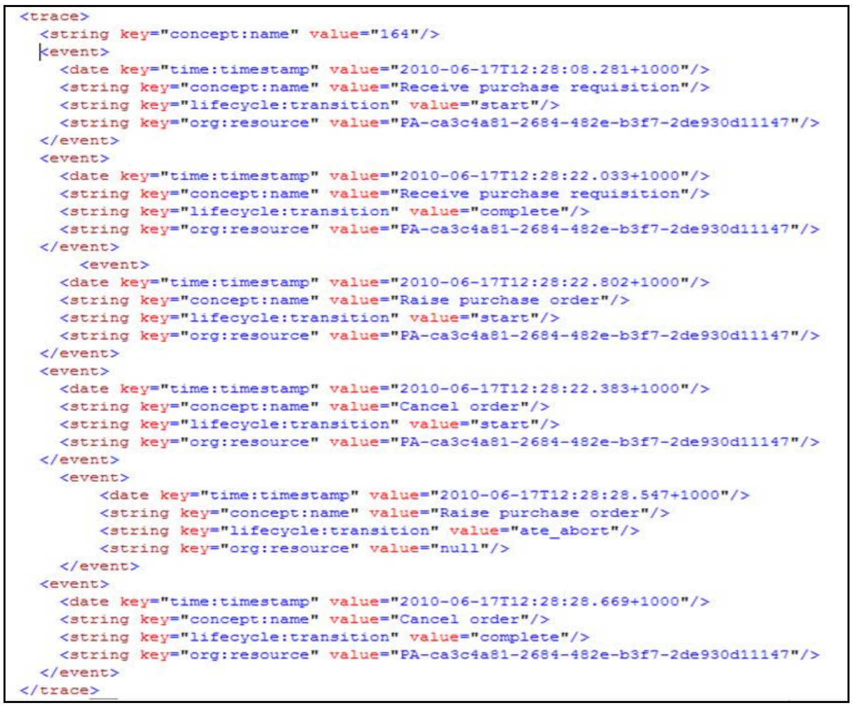
The XES log and YAWL process specification can be imported into ProM 6 through their corresponding import plug-ins. These import plug-ins ensure the selected objects are imported using the right method, and are stored in a valid format. The objects can then be fed to the YAWL process log replayer plug-in, which, after configuration, outputs three new objects: Performance metrics, YAWL replay results and Graph animation (see Figures 3 and 4). Performance Metrics contain general performance information about the entire YAWL net, the YAWL Replay Result stores the visualisation in the form of a YAWL Performance Diagram (YPD), and the Graph Animation shows an animation of replaying each trace of the imported XES log file in the YAWL specification.

An excerpt from the event log for the *Purchase Order* process in XES format can be found in Figure 5. An XES process log contains several traces (each representing a case or process instance), while each trace contains a number of events (audit trail entries). In Figure 5, six events of a particular trace are shown. The start and complete events of the *Receive purchase requisition* task are logged, as well as the start events of the succeeding tasks, *Raise purchase order* and *Cancel order*. In this case, the complete event of the Cancel order task is logged and the logs show that the *Raise purchase order* task is cancelled as a result (*ate_abort* event).

What makes the YAWL process log replayer unique is its support for complex constructs like multiple instances, cancellation regions and OR-joins. The replayer is based on the replayer for Flexible Models (Adriansyah 2009), which, while providing a solid foundation, required a number of additional constructs and techniques to handle the added complexities of the YAWL language. Unlike Flexible Models, YAWL models may contain tasks and conditions, but it is not mandatory to place conditions between tasks, which required a special consideration in visualisation. The presence of multiple instances in a process model also requires a careful analysis of the performance results. Cancellation events formed a further challenge, since cancellation can be performed at any time, either by users or by other tasks. By keeping track of different cancellation events (for tasks and cases), the YAWL log replayer provides insight into the various cancellation actions carried out during process execution.

Performance Metrics

Since the YAWL environment ensures conformance between a process specification and its event logs, our main goal of the replayer therefore was to look at ways to analyse the performance of an executed YAWL specification. The developed log replayer keeps track of various performance metrics when performing the replay, such as average throughput time (for both the entire net and individual tasks), the number of cases completed for the specification, completed events for tasks, as well as metrics related to cancellation, such as times cancelled by user or by task. This information is then used in the visualisation of the performance metrics.



```
<trace>
  <string key="concept:name" value="164"/>
  <event>
    <date key="time:timestamp" value="2010-06-17T12:28:08.281+1000"/>
    <string key="concept:name" value="Receive purchase requisition"/>
    <string key="lifecycle:transition" value="start"/>
    <string key="org:resource" value="PA-ca3c4a81-2684-482e-b3f7-2de930d11147"/>
  </event>
  <event>
    <date key="time:timestamp" value="2010-06-17T12:28:22.033+1000"/>
    <string key="concept:name" value="Receive purchase requisition"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="org:resource" value="PA-ca3c4a81-2684-482e-b3f7-2de930d11147"/>
  </event>
      <event>
    <date key="time:timestamp" value="2010-06-17T12:28:22.802+1000"/>
    <string key="concept:name" value="Raise purchase order"/>
    <string key="lifecycle:transition" value="start"/>
    <string key="org:resource" value="PA-ca3c4a81-2684-482e-b3f7-2de930d11147"/>
  </event>
  <event>
    <date key="time:timestamp" value="2010-06-17T12:28:22.383+1000"/>
    <string key="concept:name" value="Cancel order"/>
    <string key="lifecycle:transition" value="start"/>
    <string key="org:resource" value="PA-ca3c4a81-2684-482e-b3f7-2de930d11147"/>
  </event>
    <event>
        <date key="time:timestamp" value="2010-06-17T12:28:28.547+1000"/>
        <string key="concept:name" value="Raise purchase order"/>
        <string key="lifecycle:transition" value="ate_abort"/>
        <string key="org:resource" value="null"/>
    </event>
  <event>
    <date key="time:timestamp" value="2010-06-17T12:28:28.669+1000"/>
    <string key="concept:name" value="Cancel order"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="org:resource" value="PA-ca3c4a81-2684-482e-b3f7-2de930d11147"/>
  </event>
</trace>
```

Figure 5: A fragment of the Purchase Order XES log

The resulting metrics calculated can be divided into different categories, those for the YAWL Net in general and those for its individual elements (i.e. tasks, flows and conditions):

- **Net**: stores performance metrics that are related to the entire net. Currently the *Average Throughput Time and* the *Number of Completed Cases* for the net are stored.
- **Tasks:** The most interesting metrics are those calculated for tasks:
  - *Average Throughput Time*: the average throughput time for the task, only considering completed events.
  - *Number of Completed Events*: the number of times the task both starts and completes.
  - *Times Cancelled By User*: the number of times a running case is cancelled by the user while the task is in a state of execution.
  - *Times Cancelled By Other Tasks:* the number of times this task was cancelled by another task.
  - *Times Cancelled Out Other Active Nodes*: the number of times the task's cancellation region is executed.
- **Flows:** A flow in YAWL refers to the directed arc between two tasks, or between a task and a condition. Performance information for flows contain the following:
  - *Moving Time*: the traversal time between a *complete event* of the flow's input node and the *start event* of the flow's output node.
  - *Frequency:* the number of cases flowing through the flow.
  - *Times Cancelled by Task:* A flow between two tasks contains a so-called implicit condition — a condition that is not explicitly defined by the process designer but nevertheless exists internally. If a completing task has a flow in its cancellation set, any tokens within the flow's implicit condition are removed from the net.
- **Conditions:** The performance metric that is calculated for conditions (start and end conditions are special types and so are ignored) concerns the number of times it was *cancelled by another task.* That is, if the condition is contained in a task's cancellation set, and contains tokens, upon execution of that task all tokens are removed and the frequency is incremented accordingly.

This existing set of performance metrics can be easily extended by collating more event data while performing a replay. The data that is necessary to calculate the most common metrics can be found through the event log.

Visualisation

Visualisation of the replayed logs is achieved through the production of YAWL Performance Diagrams, which are based on Fuzzy Performance Diagrams (FPD) (Adriansyah et al. 2009). An FPD is a graph with performance metrics calculated from replaying a log file in a Simple Precedence Diagram projected on it, making them visually very powerful, allowing the user to quickly detect bottlenecks in a specification. These diagrams were extended to add support for the YAWL models, since FPDs only capture one type of node (i.e. tasks). The resulting YAWL Performance Diagram (YPD) also contains conditions and metrics related to cancellation events. Figure 6 shows the overall graph resulting from executing the YAWL log replayer on the Purchase Order example (note the size differences in the nodes). Detail of selected YPD nodes from Figure 6 can be seen in Figure 7.

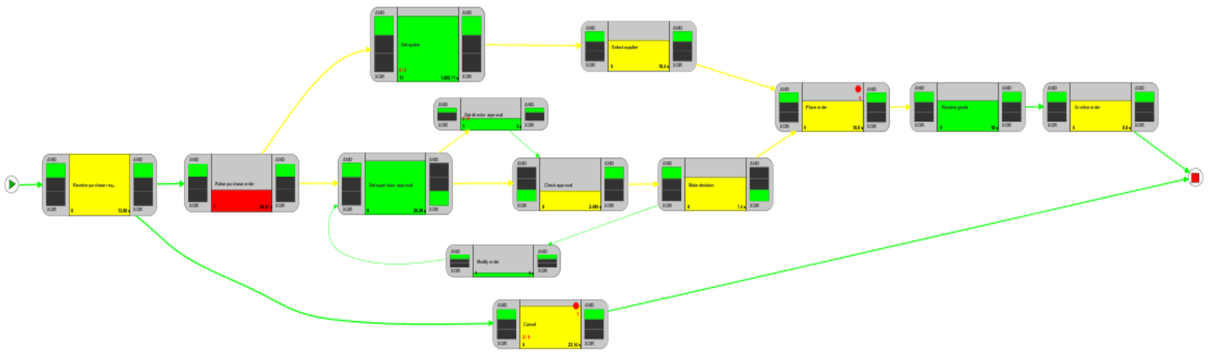


Figure 6: Screenshot of the YAWL Process Log Replayer Result

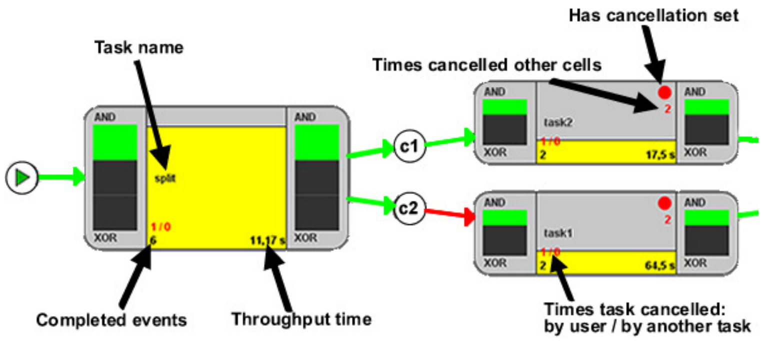
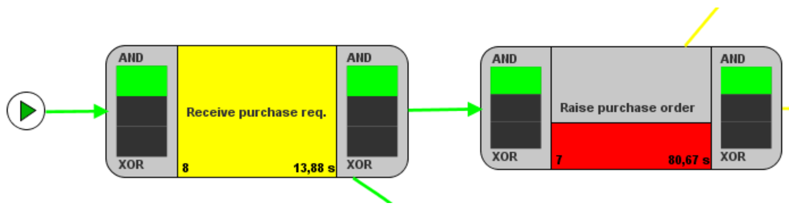


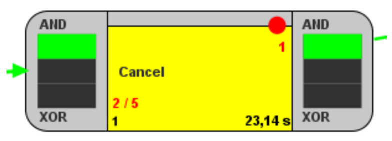
Figure 7: An Explanation of YAWL Performance Diagrams

Before we discuss the exemplary results for the *Purchase Order* example, we first describe the information visualised by the YPD nodes (see Figure 7). When comparing a YAWL specification with the resulting performance diagram, it can be seen that conditions and flows are more or less left unchanged and are incorporated directly. Each YAWL task is mapped to a YPD node and its split and join decorators are also shown. The labels of conditions and flows contain information about cancellation, if applicable. The same applies for tasks: additional information can be read from the label, for example the frequency of and times cancelled by another task.

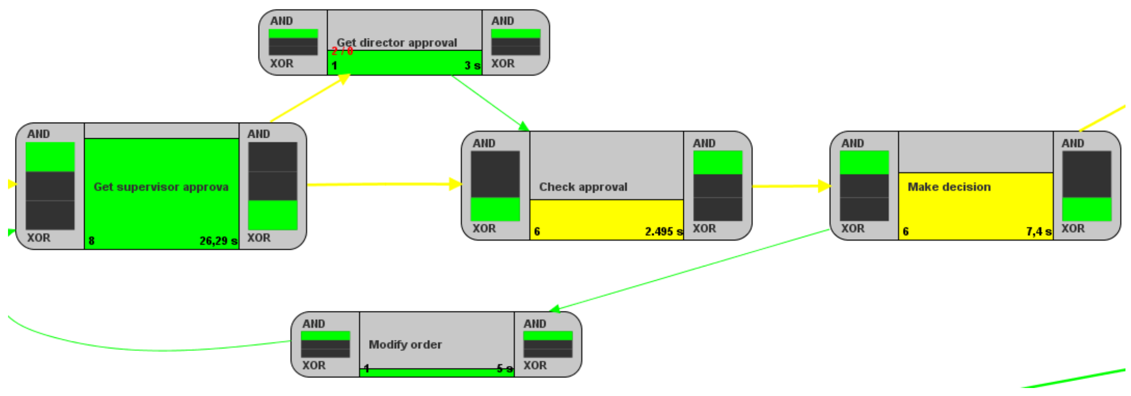
The performance metrics for each task is shown in the corresponding YPD node (i.e., the number of completed events, throughput time, the number of time a task is cancelled, etc.). Besides the metrics that can be read directly from the graph, a YPD node utilises *size and colours* to provide additional information about the performance of a process. The height of a YPD node indicates the frequency of process instances that include the executions of that task; the more instances that execute a particular task, the larger the node. Since the number of times a task is executed is related to its importance in the process, this feature visually identifies key tasks. Other performance indicators of a task use colour to alleviate information overload. The height of the coloured box inside the task shows the ratio of cases in which the task occurs compared to all cases in the event log. The colour of the box (green, yellow, or red) indicates the average throughput time of the task compared to its lower and upper bounds. A red colour indicates that the average throughput time is relatively high, a yellow colour indicates that the value is moderate, while a green colour indicates it is relatively low. The colouring of tasks in this way provides clear visual cues to possible hotspots and bottlenecks. Information about joins and splits is derived from the YAWL specification; a filled (green) upper box signifies an AND-join (or split), a filled middle box an OR-join (or split) and a filled lower box an XOR-join (or split). A flow in a YPD has indicators for frequency and performance as well. Flow thickness represents how often cases were routed from the source node to the target node; these nodes can be either tasks or conditions. The colour of a flow indicates whether the average time spent on it is relatively high (red), medium (yellow) or low (green) compared to its own upper and lower bounds.



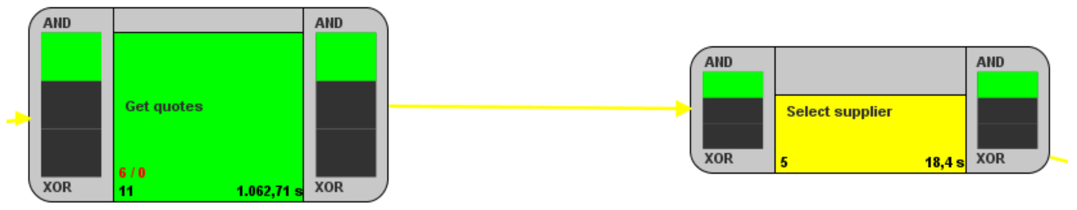
(a) Task *Receive purchase requisition* and *Raise purchase order*



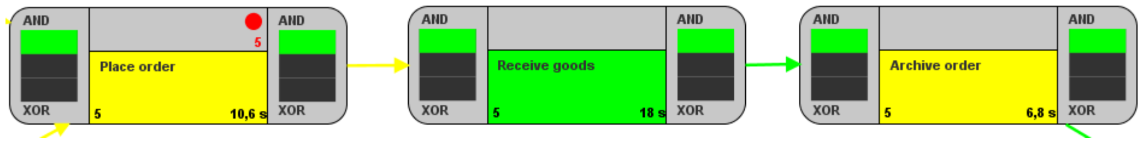
(b) Task *Cancel order*



(c) Subprocess *Get approval* and task *Make decision*



(d) MI task *Get quotes* and task *Select supplier*



(e) Task *Place order*, *Receive goods* and *Archive order*

Figure 8: Detailed overview of the YPD nodes in Figure 6

The distribution between the number of green and yellow flows is more or less equal. Thick yellow flows might be worth paying attention to and should be investigated further, for example, the flow between the *Raise purchase order* to *Get quotes* tasks. This indicates that after completion of the *Raise purchase order* task, it takes a long time before quotes are requested. This might be due to the fact that new supplier details need to be sought, or that the person responsible for this task is unable at the time. Besides the visualisation, our replayer returns two metrics that hold for the whole net as well: the number of completed cases equals *five*; and the average throughput time for these five cases equalled *2 hours and 39.8 minutes*.

RELATED WORK

The field of replaying event logs in process models has been studied before. However, the focus has mainly been on verifying the conformance of the log to the model. An overview of the work in the area of conformance is given in Rozinat et al. 2008. There, given a Petri net and a log, various conformance metrics are calculated using replay analysis, i.e., by firing transitions of the Petri net, when corresponding events occur in the log. Unfortunately, these replay approaches cannot deal with more complex routing constructs, such as the OR-split and OR-joins and therefore they are not directly applicable to YAWL models.

In Günther's thesis (2009), a replay approach is presented for a class of models, called fuzzy models, without the strict semantics of Petri nets. Again, conformance information is computed, but no performance information is provided. Furthermore, due to the very relaxed semantics of fuzzy models, results coming from the replay analysis are very hard to interpret. Currently, very few techniques are available to project performance related information onto discovered process models.

A comparison of commercial process monitoring tools in Hornix (2007) showed that (1) performance values are either measured with the requirement of having a user-defined process model directly linking events in the log to parts of the model or (2) they are measured totally independently from process models. One of the few industrial replayers available is present in BPM|One, a web-based tool offered by Pallas Athena[5]. However, this replayer doesn't consider complex constructs.

An exception is the work presented in van der Aalst et al. (2002) where performance indicators are derived from timed workflow logs. Before the performance measures are calculated, a process model in the form of a coloured workflow net is extracted from the logs using a mining algorithm. Then, the logs are replayed in the resulting net to derive performance measurements. Unfortunately, this approach relies on the discovered model, not a given process specification, to fit the log, i.e., each case in the log should be a trace in the discovered Petri net. For complex or less-structured logs, this often results in a "spaghetti-like" net, showing all details without distinguishing what is important and what is not. Hence, it is difficult for process owners to obtain any useful insights out of these models.

The work presented in this paper is based on the work of Adriansyah et al. (Adriansyah 2009; van Dongen 2009). In this work, a replay technique using flexible models was proposed. These models do not have executable semantics, but their semantics are only known if a case was finished and logged. This approach is therefore robust to different control-flow constructs and hence applicable to processes with complex constructs. Nonetheless, some extensions have been made, in order to cope with all control flow constructs that YAWL supports in particular, the support for cancellation events in the replayer as discussed previously. Since the YAWL environment ensures conformance between a process specification and its event logs, the YAWL process log replayer therefore focuses on the performance analysis of an executed YAWL specification.

CONCLUSION

The paper proposes an approach to determining the performance metrics of a business process specification given a corresponding event log. We presented the resulting process log replayer framework, which makes use of the Business Process Automation environment, YAWL, and the process mining framework, ProM. The resulting performance metrics are visualised using Performance Diagrams, which have the calculated performance information projected on them. We illustrated the performance metrics of a YAWL process model that makes use of complex constructs such as loops, nested processes, multiple instances and cancellation regions. Since conformance of the event log and the specification are automatically ensured by YAWL, each trace of the event log is replayed properly and performance metrics are tracked.

The current version of the replayer mainly focuses on the control flow of a YAWL process model, therefore, various extensions and improvements are possible. Its functionality can be further expanded by taking the resource and data perspectives into account. YAWL process models may contain complex resource settings, and replaying an event log from a simulation of this model may produce new insights into process-resource interactions, such as how often are tasks piled onto a particular resource, reoffered to other resources, executed by resources from another resource group, throughput achieved by various resources and so on. From the data perspective, more complex analysis could be done on process-data interactions, such as how data values influence the process flow. Besides these two aspects, additional performance metrics could be calculated by our replayer, such as synchronization and waiting time for tasks, the mean execution time prior to a cancellation and so on. Sophisticated graph animations based on the resulting performance metrics could also be developed. While an example log from the YAWL environment is used to demonstrate the implemented approach, the benefits it provides are in no way limited to that environment (Adriansyah 2009, Günther 2009).

[5] http://www.pallas-athena.com/products/bpmone

REFERENCES

van der Aalst ,W.M.P., and van Dongen, B.F. 2002. "Discovering Workflow Performance Models from Timed Logs." In S. Tai Y. Han and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems*, volume 2480 of LNCS, pp. 45–63. Springer-Verlag.

van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., and Barros, A.P. 2003. "Workflow Patterns." *Distributed and Parallel Databases*, 14(3):5-51.

van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A.J.M.M. 2003. "Workflow Mining: a Survey of Issues and Approaches." *Data and Knowledge Engineering* (47:2), pp. 237–267.

van der Aalst, W.M.P., Weijters, A.J.M.M., and Maruster, L. 2004. "Workflow mining: Discovering Process Models from Event Logs." *IEEE Transactions on Knowledge and Data Engineering,* 16(9):1128–1142.

van der Aalst, W.M.P., and ter Hofstede, A.H.M. 2005. "YAWL: Yet Another Workflow Language." *Information Systems*, 30(4):245-275.

van der Aalst, W.M.P., van den Brand, P.C.W., van Dongen, B.F., Günther, C.W., Mans, R.S., Alves de Medeiros, A.K., Rozinat, A., Song, M., Verbeek, H.M.W., and Weijters, A.J.M.M. 2007. "Business Process Analysis with ProM." *In Proc. of the 17th Annual Workshop on Information Technologies and Systems*, pp. 223-224.

Ardhaldjian, R., and Fahner, M. 1994. "Using Simulation in the Business Process Reengineering effort". *Industrial engineering*, July, pp. 60–61.

Adriansyah, A. 2009. *Performance Analysis of Business Processes from Event Logs and Given Process Models*. Master's Thesis, Eindhoven University of Technology, The Netherlands.

Adriansyah, A., Dongen, B.F., and van der Aalst, W.M.P. 2010. "Towards Robust Conformance Checking", *International Workshop on Business Process Intelligence BPI 2010*, Hoboken, New Jersey.

Brataas, G., Hughes, P., and Sølvberg, A., 1997. "Performance Engineering of Human and Computerised Workflows." *9th International Conference on Advanced Information Systems Engineering*, pp. 187-202, Springer.

van Dongen, B.F. and Adriansyah, A. 2009. "Process mining: Fuzzy clustering and performance visualization." In *Business Process Management Workshops*, volume 43, pp. 158–169. Springer.

Günther, C.W. 2009. *Process Mining in Flexible Environments*. PhD Thesis, Eindhoven University of Technology, The Netherlands.

Günther, C.W. 2010. "OpenXES Developer Guide." Retrieved 15 April 2010, from http://code.deckfour.org/xes

ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., and Russell, N. (editors) 2010. *Modern Business Process Automation: YAWL and its Support Environment.* Springer.

Hornix, P.T.G. 2007. *Performance Analysis of Business Processes through Process Mining*. Master's Thesis, Eindhoven University of Technology, The Netherlands.

Jablonski, S., and Bussler, C. 1996. *Workflow Management: Modeling Concepts, Architecture and Implementation.* International Thomson Computer Press.

Rozinat, A., Alves de Medeiros, A.K., Günther, C.W., Weijters, A.J.M.M., and van der Aalst, W.M.P. 2008. "The Need for a Process Mining Evaluation Framework in Research and Practice." *In* A. ter Hofstede, B. Benatallah, and H.Y. Paik, editors, *Business Process Management Workshops*, volume 4928 of LNCS, pp. 84–89. Springer.

Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., and van der Aalst, W.M.P. 2010. "ProM 6: The Process Mining Toolkit" *In Proceedings of BPM Demonstration Track 2010*, CEUR 615, pp. 34-39. http://ceur-ws.org/Vol-615/paper13.pdf

Weijters, A.J.M.M., van der Aalst, W.M.P., van Dongen, B.F., Günther, C., Mans, R., Alves de Medeiros, A. K., Rozinat, A., Song, M., and Verbeek, H.M.W. 2007. "Process Mining with ProM." *In Proceedings of the 19th Belgium-Netherlands Conference on Artificial Intelligence*, Utrecht, The Netherlands.

Weske. M. 2007. *Business Process Management: Concepts, Languages, Architectures*. Springer.

Wynn, M.T., Edmond, D., van der Aalst, W.M.P., and ter Hofstede, A.H.M. 2009. "Synchronization and Cancelation in Workflows based on Reset Nets", *International Journal of Cooperative Information Systems* (IJCIS), vol 18(1), pp. 63-114, 2009, World Scientific Publishing.

COPYRIGHT