The importance of actively managing and analyzing business processes is acknowledged more than ever in organizations nowadays. Business processes form an essential part of an organization, and their application areas are manifold. Most organizations 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, analyzes, and visualizes a variety of performance metrics using a process definition and its execution logs. Performing performance analysis on existing and planned process models offers a great way for organizations to detect bottlenecks within their processes and allow them to make more effective process improvement decisions. Our technique is applied to processes modeled in the YAWL language. Execution logs of process instances are compared against the corresponding YAWL process model and replayed in a robust manner, taking into account any noise in the logs. Finally, performance characteristics, obtained from replaying the log in the model, are projected onto the model.

Keywords: business process management, performance analysis, process mining, Yet Another Workflow Language (YAWL), log replay analysis

Jan Mendling and Jan Recker were the Senior Editors for this paper.
INTRODUCTION

Business processes form the heart of every organization, whether small or large, and a number of processes can always be identified within an organization and their information systems. Organizations regularly undertake process improvement activities to ensure that their operational processes are as effective and efficient as possible (Weske 2007). In doing so, identification of both frequently occurring flaws and performance bottlenecks in existing process executions are essential, as they typically become the starting point of any optimization efforts (Bentley and Davis 2009; Persse 2006). Furthermore, carrying out performance analysis on existing and planned process models offers a great way for organizations to detect issues within their processes. For instance, Brataas et al. (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 analyzes of (TO-BE) processes (Ardhaladjian and Fahnre 1994). One of the drawbacks of 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 can be very time-consuming (Reijers and van der Aalst 1999). Simulation requires a model which reflects the behavior of a process, including the data and resource perspectives (Wynn et al. 2008). Traditionally, making such models requires a collaborative effort between key stakeholders and process analysts (Poulymenopoulou et al. 2003), which implies lengthy discussions with workers, extensive document analysis, and careful observation of participants.

Executable process models, i.e., workflows, contain descriptions of what tasks need to be performed, when they need to be performed, by whom they need to be performed, what information they need, and what information they produce. Workflows exist independently of workflow management systems that provide process executions, i.e., a workflow in a system may exist merely as a guideline, without enforcing a process execution description (van der Aalst 1998). A Process-Aware Information System (PAIS) is a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models (Dumas et al. 2005). Most PAISs today provide some kind of event log (also referred to as a transaction log or audittrail) (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 event logs, a better alternative to process simulation is to make use of such log data for performance analysis.

Ideally, the executions of cases captured in these logs correspond one-to-one to the given model. However, practical situations often require that users deviate from the model in their executions, e.g., people often make mistakes which need to be corrected by system administrators, etc. Also, a PAIS system can encompass more than a workflow management system, working together with many other systems (e.g., Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) systems, project management tools), hence the events in the log may not correspond one to one with the activities in the predefined model. Real-life cases have shown that AS-IS process executions often deviate from their predefined process models, even in processes that are supported by information systems (Rozinat and van der Aalst 2008; Rozinat et al. 2009; Adams et al. 2006). As the execution of activities in the AS-IS system may not be strictly enforced by the model, the performance analysis technique must cater to possible inconsistencies between the event log and the model.

CONTRIBUTION

This paper provides a robust replay technique that computes performance characteristics using process data collected while it is being executed. The approach presented in this paper supports process models with advanced constructs such as loops, nested processes, multiple instances and cancellation regions. These processes are modeled using the YAWL language and executed in the YAWL open-source business process automation environment. We have developed a sophisticated process log replayer framework, as part of the open-source process mining framework, ProM. The replayer software allows us to calculate the performance metrics of a complex YAWL process by replaying logs against a given model. While replaying, the replayer checks to what extent logs indeed conform to a YAWL process description, minimizes the deviations and analyzes the information that is contained in the log. The resulting performance metrics from AS-IS log data are visualized using Performance Diagrams, which have the calculated performance information projected onto the model.

The contributions of the paper would be of interest to business process management professionals who have the need to analyze performance metrics of complex process models with a view to better understand the conformance of existing systems (evidenced by the log data) to either AS-IS or TO-BE process models.
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 synchronization. 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 synchronization 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). However, many of the existing modeling languages and tools are unable to express such complex concepts (van der Aalst et al. 2005).

Our main objective is to propose a performance analysis technique that (1) can support complex process models with advanced constructs such as cancellation, multiple concurrent instances, and advanced synchronization as identified by the work on workflow patterns; and (2) can cater for deviations detected in the logs when the log data is compared against a process model (i.e., a log and its corresponding process model is not fully compliant).

This paper demonstrates how the performance analysis of complex process models can be carried out with the use of an event log replayer, a tool that will replay all recorded process executions of a process in an event log against a model of the process to calculate and store metrics related to performance analysis. Specifically, the paper focuses on the design and the development of an event log replayer for processes modeled using the Yet Another Workflow Language (YAWL) notation (van der Aalst and ter Hofstede 2005), which is a reference implementation of the workflow patterns, and the visualization of the calculated performance metrics. We extend the earlier work of Piessens et al. (2010) by removing the explicit assumption that the log and the YAWL model are fully compliant. We focus on the situation in which an event log is produced by an AS-IS system, while the model describes either the AS-IS situation or the desired situation (i.e., TO-BE model).

There are several potential benefits that can be gained from utilizing such a replay technique. First, the visualization of performance metrics provides an organization with a better understanding of its business operations represented by complex process models. Second, the ability to identify performance bottlenecks of current business processes is a very useful starting point to optimize AS-IS models and for resource planning. Third, the use of replay techniques on TO-BE models can provide more accurate performance metrics than otherwise might be possible from simulation experiments. Finally, the ability to analyze and detect possible deviations between event logs and a given complex process model makes the proposed replay technique suitable for real-life event logs where such situations commonly occur.

The remainder of the paper is organized as follows: first, we describe some background to the approach taken, together with a purchase order process modeled 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 and results. We also illustrate the performance analysis results based on the purchase order example. Before concluding the paper, related work is discussed.

BACKGROUND

Process mining is a technology that uses event logs (i.e., recorded actual behavior) to analyze 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 utilization, 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 a process definition, such as “hotspots” or bottlenecks that cannot be identified by mere inspection of the static model alone. A much better insight into the root causes of bottlenecks in the existing AS-IS process models can be obtained when we use those models for comparative analysis with event logs from the AS-IS system (i.e., log replay). In addition, we can make use of replay techniques to perform comparative analysis with TO-BE models, in a similar manner to carrying out simulation experiments during process improvement phase.

Event log replayers are tools developed to replay, or even simulate the event log of a process. In reality, process instances, or cases, are executed independently from each other. For example, in an insurance company, the way a claim is handled does not directly influence how other claims are handled. Therefore, replay techniques treat events that belong to a particular case independently from others. Events of a case are often referred to as the trace of the case. While replaying, the replayer checks to what extent traces are indeed executable in a given process model, minimizes the deviations (where necessary) and analyzes 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 deviations, 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), which offers a range of log replayers, each designed for a specific type of process model, such as Petri nets (Rozinat et al. 2008) or Flexible models (Adriansyah et al. 2010). However, none of the existing log replayers can handle processes that contain more complex control flow dependencies, such as cancellation, multiple concurrent instances, and advanced synchronization. For example, the replay technique in Rozinat and van der Aalst (2008) requires process models to be in the form of classical Petri nets, and classical Petri nets cannot express some of the more complex concepts. Another replay technique, proposed in Günther (2009), assumes models to be in form of Fuzzy models, which also cannot express cancelation. Thus, these existing replay techniques are insufficient to carry out performance analysis of process models containing such complex constructs.

The YAWL language was created as the reference process modeling 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 other languages, such as Business Process Modeling Notation (BPMN) and Event-Driven Process Chains (EPCs). The system also provides support for execution of complex control flows with sophisticated resource allocation mechanisms. As such, YAWL provides an ideal environment to model and execute complex processes and to generate detailed event logs during each execution. Hence, we demonstrate how log replay techniques can support such advanced concepts using processes modeled in the YAWL language.

To demonstrate how business processes can be modeled in YAWL, we present a fictitious purchase order process modeled 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.

A YAWL process has a unique starting point (input condition) and a unique endpoint (output condition). Tasks (activities) in the model are represented by rectangles (e.g., the Receive purchase requisition task), with all tasks occurring 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 behaviors of a task are modeled 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 of a purchase order process.](http://www.yawlfoundation.org)

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 (modeled 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

---


2 http://www.yawlfoundation.org.
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 this example process 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 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 region associated with it. These cancellation regions may contain any number of other tasks and/or conditions in the same net, which will be cancelled 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 (see Figure 2).

![Figure 2: YAWL purchase order process from Figure 1 with cancellation regions for tasks shown.](image)

**ROBUST REPLAY TECHNIQUES**

In this section, we first outline the requirements for a replay technique that supports the projection of performance information onto a process model, given an event log and the model. It is then followed by a detailed description of the proposed replay approach, including an illustration based on the YAWL purchase order example.

As mentioned earlier, many existing process modeling languages are unable to express complex concepts, although they are often needed to successfully model real-world processes. Therefore, we nominate the ability to handle complex constructs appropriately as the first requirement of a robust replay technique. To our knowledge, existing replay techniques require models to be in specific modeling languages, such as Petri nets or Fuzzy models, that may not be able to express the complex concepts supported by workflow patterns. As YAWL is one of the modeling languages that can express complex concepts, in Piessens et al. (2010) we presented a technique to project performance information onto a YAWL model, under the assumption that the event log used to obtain the performance information originated from the YAWL system using exactly that YAWL model. In practice, the information needed to obtain performance information is stored in several different systems and the process model may describe another situation rather than the one prescribed. As the currently existing replay technique on YAWL requires perfectly corresponding logs, its applicability is limited when it comes to real-world logs generated by other systems.

One of the advantages of analyzing performance using process models projected with performance information is that the root cause of a performance problem can be traced according to the semantics of the model. However, if the projected performance information is retrieved from a log that does not fit a model, conclusions that are drawn from
it may be misleading. For example, suppose that we have a model and a log that does not fit the model. The model
describes a process that always starts with activity A, followed by an optional execution of sequence of activities
BCD, and ends with activity E. Thus, the set of traces that perfectly fit the model would be \{ABCDE,AE\}. The log
consists of only two non-fitting traces, ACDE and ABE. Projection of performance information onto the model will
show that the number of occurrences of activity B, C, and D is the same (i.e., each occurs once). Since according to
the semantics of the model, BCD is in a sequence, one may draw a false conclusion from the projected information
that there is a single trace in the log where sequence BCD occurs once during trace execution. Such a false
conclusion may lead to other false analyses. Therefore, projected performance information needs to be consistent
with the semantics of the model.

Given a model and an unfitting trace from a log, the quality of the projected performance information depends on the
number of modifications performed to make the original traces perfectly fit the semantics of the model (i.e., the
fitness of the original traces to the model). To preserve the quality of projected information, the number of
modifications needs to be kept to a minimum. The higher the fitness values of the traces to the model, the lesser the
modification required to the traces to make them fit. Therefore, it is crucial to identify each addition and removal
operation required to make the trace fit perfectly and to preserve the original activities as much as possible.
Therefore, we nominate this as the second requirement of a robust replay technique. For cases where the log is
not fully compliant with the process model, for each identified deviation, a replay technique should be able to identify
what should have occurred using a proper reasoning without manual interference (e.g., which activities in the model
are skipped in reality, which extra activities are recorded in event log but should not occur according to the model),
such that traces in the log can be “massaged” to be perfectly fit with minimal modifications.

![Diagram](https://example.com/diagram.png)

**Figure 3: Overview of the proposed replay approach.**

Based on these two requirements, we chose the replay technique that is based on the A* algorithm proposed in
Adriansyah et al. (2010, 2011). Given a model and a trace, the algorithm explores the state space of the model to
identify deviations between (1) the model and the trace and (2) what should have occurred. An overview of our
approach is shown in Figure 3. We start from a YAWL model that describes either the AS-IS condition or TO-BE
situation, and an AS-IS event log. Then, we transform the YAWL model to a Flexible model (Adriansyah et al. 2010)
with cancellation region extension, taking into account information about observed activities in the log. Next we
replay the log against the Flexible model, using a replacer specifically developed to handle cancellation. The result is
an augmented log, completely compliant with the original YAWL specification, and using this log we compute performance characteristics and project them onto the YAWL model.

The proposed replay technique works based on directed state space exploration, in which the state space universe is defined from the model, log, and their combination. Since we can construct the state space of any YAWL model (although some might be possibly infinite), the technique is applicable. The output of the replay can be used to explicitly pinpoint causes of deviations, as shown in Adriansyah et al. (2011). Furthermore, the replay result is optimized according to a cost function that can be defined in such a way that the number of deviating activities is minimized.

There were also conceptual considerations behind the selection of Flexible models. Any behavior that is expressed by a YAWL model with complex constructs, i.e., cancellation region, multiple instance, and composite tasks, is expressible by an isomorphic Flexible model, while, with other modeling languages, the required transformation to show the same expression may be more complex. For instance, the OR-split and OR-join semantics of a YAWL node in a YAWL model can be isomorphically represented as a node in a Flexible model. Expressing the same semantics in classical Petri net, for example, would require a transition, and the addition of a set of invisible transitions and a set of places. Due to the isomorphism, mappings from the original model to replay results of Flexible models and back to YAWL models are straightforward. Furthermore, a replay technique for Flexible models is already defined in Adriansyah et al. (2009), and later shown to be extendable in Adriansyah et al. (2011). In a similar way, we can extend the replay technique to consider Flexible models with cancellation regions.

It is worth mentioning that other than the one based on A*, there is no other approach known to us that satisfies all requirements. The approach proposed by Rozinat and van der Aalst (2009) requires manual reasoning for each occurring deviation and, thus, does not minimize the number of deviating activities. An approach by Cook et al. (2001) is slightly similar to the one in Adriansyah et al. (2011), except that it does not guarantee minimum number of identified deviations due to heuristics. Furthermore, the latter also unnecessarily adds extra activities if proper termination of the process is not reached.

To illustrate our technique, we now use the YAWL example introduced in the previous section, together with the event log described in Table 1, which contains some example events recorded for a particular requisition order. The entries in Table 1 show that the order was received, after which two quotes were requested by Pete, then Bill chose a supplier and ordered the goods, which were received one month after the order began. In replay, a requisition order would be called a case and each line in the table an event. Events refer to an activity, a lifecycle transition such as start, complete, suspend, resume, etc., as well as a resource (person) that performed the activity. Furthermore, events occur at a particular point in time. Note that, for illustrative purposes, the event log shown in Table 1 has been deliberately simplified. In real-world settings, a log may contain thousands of cases, each with numerous events, and our replay technique can manage logs of any complexity. Note also that we explicitly assume that not all information is always present for all events, as can be seen in Table 1.

### Table 1: Example of a Log in an AS-IS System

<table>
<thead>
<tr>
<th>Requisition order</th>
<th>Activity</th>
<th>Lifecycle</th>
<th>Time</th>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ12235</td>
<td>Received</td>
<td></td>
<td>2 April, 2011 10:24:31</td>
<td>Pete</td>
</tr>
<tr>
<td>RQ12235</td>
<td>Request quote</td>
<td></td>
<td>2 April, 2011 10:25:31</td>
<td>Pete</td>
</tr>
<tr>
<td>RQ12235</td>
<td>Make decision</td>
<td>start</td>
<td>9 April, 2011 14:25:41</td>
<td>Bill</td>
</tr>
<tr>
<td>RQ12235</td>
<td>Make decision</td>
<td>complete</td>
<td>9 April, 2011 14:55:31</td>
<td>Bill</td>
</tr>
<tr>
<td>RQ12235</td>
<td>Receive products</td>
<td></td>
<td>1 May, 2011 9:00:00</td>
<td></td>
</tr>
</tbody>
</table>

### From YAWL to Flexible Models

Flexible models (Adriansyah et al. 2010) can be considered as an abstraction of most currently available process modeling languages, tailored toward replay analysis. In particular, these models do not have formal, executable semantics, but instead, for a given behavior, we can determine whether this behavior complies with the model and, if not, where the deviations are.

Flexible models are directed graphs in which each node represents at most one event type in the log (e.g., “purchase requisition received,” or “decision making started”). For all nodes, pre- and post-conditions are given in terms of combinations of other nodes that should occur before, or will occur later, in a case. This reasoning allows
for deciding if, for a given case, all conditions have been satisfied. Furthermore, these models allow us to decide, for a case in which not all conditions are satisfied, where deviations occur in terms of events that should have been executed in the model, but did not occur in the log, and events that show in the log, but should not have occurred according to the model. Also, the occurrences of nodes that do not correspond to events are correctly identified. This allows us to augment the data in the log in such a way that we are left only with cases corresponding fully to the YAWL model, which is a precondition for performance calculations.

Figure 4 shows the translation of the YAWL model of Figure 2 into a Flexible model. Here, we mapped the activities shown in the log and their lifecycles, i.e., the YAWL activity “receive purchase order” corresponds to the “received” event in the log, the “get quotes” activity to the event “request quote,” the “make decision” activity to the events “make decision start” and “make decision complete,” and the activity “receive goods” to the event “receive products.” For an overview of how to translate events referring to “start,” “complete,” etc., we refer to Appendix A. The other YAWL activities are represented in the Flexible model, but they are not mapped to any events.

When comparing YAWL models and Flexible models, it is clear that there are essential differences. For example, in YAWL, activities may not correspond to single event types, as shown by our example, where the activity “Make decision” is represented both by the event “Make decision start” and “Make decision complete.” Therefore, when translating YAWL models to Flexible models, we avoid introducing too many nodes that do not correspond to events. Each YAWL activity is translated into as many nodes in the Flexible model as necessary given the corresponding event types in the log. If the YAWL model contains places, then these are always translated into a “start” and an “end” node in the Flexible model in order to later simplify our performance computations.

The pre- and post-conditions of the nodes in the Flexible model are determined by the YAWL model as well. For single activities, the mapping is straightforward and can be obtained directly from the model. For more advanced constructs, however, the mapping is more involved. For instance, a multiple instance node in YAWL is mapped to a sequence of two nodes in a Flexible model, where the input node of the sequence has the same preconditions as the preconditions of the multiple instance node, and the output node has the same post conditions as the post conditions of the multiple instance node. For a complete overview of how to automatically map a YAWL model to a Flexible model, we refer to patterns in Appendix B. By first analyzing the log to find event types for each activity and then using the corresponding pattern, transformation from YAWL models to Flexible models can be performed automatically without any manual process. Furthermore, the transformed YAWL models are trace equivalent to the Flexible model, i.e., traces that are valid according to the YAWL model are also valid according to the Flexible model, and vice versa.

For this paper, we extended the notion of Flexible models (Adriansyah et al. 2010) with cancellation regions. This extension simply limits the horizon for checking whether pre- and post-conditions are satisfied, i.e., for checking the post-condition, we do not need to look further ahead than the cancelling activity, and for pre-conditions we can start looking back from the cancelling activity.

**Augmenting the event log**

After translating the YAWL model to the appropriate Flexible model and replaying the log, we identify deviations between the event log and the model. There are two possible deviations, namely, events that should have occurred according to the model, but are not found in the log (skipped activities), and events that occurred in the log, but should not have, according to the model (inserted activities). When determining deviations, we favor the
identification of skipped activities, the main reason being that the inserted activities require us to remove events to obtain a conformant log, but removing events implies removing potentially relevant information for performance measurements.

Although the replay technique identifies the location of skipped activities in the case (e.g., the event “get approval” occurred before “make decision” was started, but after the event “order received” occurred), it does not identify the moment in time when this event should have occurred. Therefore, we need to make some assumptions. Here, we assume that an event that corresponds to a YAWL activity occurs as early as possible, i.e., the chosen timestamp for the inserted event is the earliest possible, depending on which event precedes it in the case. However, for events that relate to the “end” event of places, we assume the timestamp to be the latest possible. In both cases, we do not consider the linear order of events in the case, but the partial order of events as identified by the replacer.

The result of augmenting the log of Table 1, after replaying it in the model of Figure 4 is shown in Table 2. In Table 2, the original events of Table 1 are all present. However, the replay algorithm introduced events corresponding to activities in the YAWL model that were not present in the log. Furthermore, it introduced timestamps where they were missing, based on the assumptions above. The resulting log is a fully compliant execution of the model of Figure 4, which allows us to do performance computations in a straightforward way.

<table>
<thead>
<tr>
<th>Req. order</th>
<th>Activity</th>
<th>Lifecycle</th>
<th>Time</th>
<th>Person</th>
<th>Mapped event</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ12235</td>
<td>Input Condition</td>
<td></td>
<td>2 April, 2011 10:24:31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Receive purch. Req.</td>
<td>2 April, 2011 10:24:31</td>
<td>Received</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Raise purch. order</td>
<td>2 April, 2011 10:24:31</td>
<td>Request quote</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Get quotes</td>
<td>2 April, 2011 10:25:31</td>
<td>Pete</td>
<td>Request quote</td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Get approval</td>
<td>2 April, 2011 10:24:31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Make decision start</td>
<td>9 April, 2011 14:25:41</td>
<td>Bill</td>
<td>Make decision (start)</td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Make decision complete</td>
<td>9 April, 2011 14:55:31</td>
<td>Bill</td>
<td>Make decision (complete)</td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Select supplier</td>
<td>2 April, 2011 10:25:31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Place order</td>
<td>9 April, 2011 14:55:31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ12235</td>
<td>Receive goods</td>
<td>1 May, 2011 9:00:00</td>
<td>Receive products</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other than the events in the log that are executed properly according to the replay result, the timestamps of other events in the augmented log are estimated. Therefore, the quality of our performance calculation depends linearly on the information from the original log that is still preserved in the augmented log. A fitness value between the log and the model, as defined by Adriansyah et al. (2010), would be a good indicator of the quality of performance measurement. The lower the fitness value between a log and a model, the higher the number of deviations between data in the log and the model, and thus a greater number of assumptions that need to be made for the timestamps of activities, and, therefore, the reliability of performance calculations lessens. Similarly, the higher the fitness value between a log and a model, the lesser the number of assumptions that need to be made, thus leading to higher reliability of performance calculations.

Performance Metrics

Once an event log consisting only of conformant cases has been produced, a performance analysis of the YAWL process model can occur. The replayer we developed 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, as well as metrics related to cancelation. This information is then used in the visualization of the performance metrics.

Since we have translated our YAWL specification to a Flexible model in which an activity can be represented by a number of nodes following the lifecycles depicted in Appendix A, we compute the processing times of activities based on the first and last event in any of these lifecycles. Using the example of Figure 1, only the activity “make decision” will show information on processing times, as this is the only activity for which we can measure the time between starting and completing the activity. For conditions in the YAWL net in particular, we will have throughput times as we map them to “start” and “end” events.

We divide the performance metrics computed for the YAWL specification into two categories, the first related to case executions in general and the second related to YAWL’s individual elements (i.e., tasks, flows and conditions). These metrics are motivated by performance metrics that are typically calculated by simulation techniques, as shown by Wynn et al. (2008). Details of the metrics are given in Table 3. This existing set of performance metrics
can be 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.

<table>
<thead>
<tr>
<th>Num.</th>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Throughput Time</td>
<td>The average, minimum, maximum, and standard deviation of execution of a case</td>
</tr>
<tr>
<td>2</td>
<td>Number of Cases</td>
<td>The number of cases in the log</td>
</tr>
</tbody>
</table>

**Task-related metrics**

| 1    | Throughput time                    | The average, minimum, and maximum throughput time of the task                                   |
| 2    | Number of times started             | The number of times the task is started (may be not completed)                                  |
| 3    | Number of times completed           | The number of times the task is both started and completed                                      |
| 4    | Times cancelled by other tasks      | The number of times the task was cancelled by another task                                      |
| 5    | Synchronization time               | The average, minimum, and maximum values of the period between the moment when a predecessor of the task is completed until all predecessors of the task are completed |
| 6    | Waiting time                        | The average, minimum, and maximum values of the period between the moment all predecessors of the task are completed until the task is started |

**Flow-related metrics**: A flow in YAWL refers to the directed arc between two tasks, or between a task and a condition

| 1    | Moving time                         | The average, minimum, maximum, and standard deviation of the period between the moment the flow's input node is completed and the moment its output node started |
| 2    | Frequency                            | The number of times the flow is visited during replay                                           |
| 3    | Times cancelled by other tasks      | 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 starting task has a flow in its cancellation set, any tokens within the flow's implicit condition are removed from the net. |

**Condition-related metrics**

| 1    | Waiting time                         | The average, minimum, and maximum values of the period between the moment a token enters a condition until the token is consumed by a task |
| 2    | Frequency                             | The total number of tokens that enter the condition during process executions                   |
| 3    | Times cancelled by other tasks       | The number of times the tokens in conditions are removed due to a cancellation action            |

**Visualization**

In the final step, the performance characteristics computed for the YAWL specification are visualized. Visualization of the replayed logs is achieved through the production of YAWL Performance Diagrams, which are based on Fuzzy Performance Diagrams (FPD) (van Dongen 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 and other issues in a specification. These diagrams were extended to add support for the YAWL models, since FPDs capture only one type of node (i.e., tasks). The resulting YAWL Performance Diagram (YPD) also contains conditions and metrics related to cancellation events. Figure 5 shows the overall graph resulting from executing the YAWL log replacer on the completed Purchase Order log example (note the size differences in the nodes). An explanation of YPD nodes can be seen in Figure 6.

Before we discuss the results for the Purchase Order example, we first describe the information visualized by the YPD nodes (see Figure 6). 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 node in YAWL (including conditions) is mapped to a YPD node and its split and join decorators are also shown. A condition is always mapped to a YPD node with XOR-split and XOR-join decorators. The labels of a YPD node contain information about cancellation, if applicable. Additional information can be read from the label, for example, the frequency of and times cancelled by another task. The YPD node that starts a process is colored green, while the node that ends a process is colored red.
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 canceled, etc.). Besides the metrics that can be read directly from the graph, a YPD node utilizes size and colors to provide additional information about the performance of a process. The height of a YPD node indicates the number of times task instances of the YPD are completed; the more task instances that are completed, the larger the node. Since the number of executions of a task is related to its importance in the process, this feature visually identifies key tasks. Other performance indicators of a task use color to alleviate information overload. The height of the colored box inside the task shows the ratio of cases in which the task occurs compared to all cases in the event log. The color of the box (green, yellow, or red) indicates the average throughput time of the task compared to its lower and upper bounds. A red color indicates that the average throughput time is relatively high, a yellow color indicates that the value is moderate, while a green color indicates it is relatively low. The coloring of tasks in this way provides clear visual clues to possible bottlenecks. As conditions are also translated as YPD nodes, the waiting times of conditions are shown in the same way as throughput times of tasks. Information about joins and splits is derived from the YAWL specification; a filled 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). The color of the join box and split box of a task indicates the average waiting time of the task and the average synchronization time of the task respectively compared to others. 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 color 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. In addition, we use light gray color to visualize flows and nodes that were never visited during replay.

Figure 7 to Figure 9 show details of the YPD nodes in Figure 5. From the high distribution of green-colored YPD nodes, we can conclude that the average throughput time of most activities is low. This includes activity “Received” that is logged without lifecycle information or timestamps, as we can still derive its performance based on other activities and the given process model. From its node color and size, we can immediately see that despite its rare occurrence, the activity “Archive order” causes bottleneck whenever it occurs (see Figure 9). In addition, although the average throughput time of activity “Make decision” is not as high as the throughput time of the activity “Archive order,” the activity is involved in all process instances (shown by the height of yellow-colored inner box). To increase performance of the process, improvement efforts would focus on these two activities, rather than an activity such as “Get quotes” that occurs often but has low average throughput time.
There are some thick arcs that are colored red (see Figure 8). These arcs are also considerably important for performance analysis, as they indicate the time spent between tasks. For example, the thick red arc between node “Get approval” and “Make decision” means that the time spent from the moment an order is approved (i.e., the moment “Get approval” activity is finished) until a decision start is made is relatively high. This may indicate that the person who makes decisions is delaying work, or there are not enough resources to work on the activity “Make decision.” Note that this problem is also indicated by the red color of waiting time of the node “Make decision.” This information can be quite useful for resource planning decisions.

In addition to the visualization, our replayer returns several performance metrics, as described in the Performance Metric section. For this example, we use ten cases with average case throughput time of 24.189 days. The minimum, maximum, and standard deviation case throughput times are 4.125 days, 29.941 days, and 48.188 seconds respectively. These values indicate that although the values of case throughput time vary, they are evenly distributed. The average throughput time of “Make Decision” and “Archive order” activities are 28.313 minutes and 24 hours respectively. Compared to the average throughput time of other activities that are equal to 0.00 (as the “start” and “complete” lifecycle is not logged), the two activities are clearly the ones that take the longest time to finish. The average flow time from the completion of “Get approval” activity to the start of activity “Make decision” is 6.021 days, and the average waiting time of “Make decision” activity is 6.362 days. From these two values, we know that in average there are six days of delay between the moment a purchase order is approved until decision making activity is started. Process experts can make use of all of these values to identify hotspots and bottlenecks of the process and use them to improve its performance.

IMPLEMENTATION
The YAWL process log replayer has been developed in ProM (Verbeek et al. 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 2.2 environment and ProM version 6 components can be found in Figure 10.

**Architecture**

The work presented in this paper has been implemented as a plug-in called “Replay log on YAWL for performance”. It can be found in the “Replayer” package, downloadable from the ProM repository 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 10. 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. 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.

![architecture_diagram](image_url)

**Figure 10: Architecture.**

The richest form of event logs supported by ProM 6 is that 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 more easily be extended to calculate metrics on other attributes than the ones we used. However, for this paper, we focused on the data that is typically provided by YAWL.

YAWL stores its process logs in a number of relational database tables, but it provides an API 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, a merged log functionality is provided; it combines related log information from both the Engine and Resource Service into a single XES 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 plugs, which 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 two new objects: a Flexible model of the YAWL specification and the performance analysis result (see Figure 10). The performance analysis result contains general performance information about the entire YAWL net. Together with the Flexible model of the specification, the analysis result is visualized in the form of a YAWL Performance Diagram (YPD).

---

3 http://prom.win.tue.nl/ProM/packages/packages.xml.
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 12, seven events of a particular non-conformant trace are shown. It can be seen that the start and complete events of the “Make decision” activity are logged, while the lifecycle of activity “Receive purchase requisition” is not recorded. Notice that the trace is not conformant, as “Raise purchase order” activity is skipped and other activities, such as “Make decision” and “Receive products,” are still executed after execution of activity “Cancel order.” However, our replay technique manages to identify skipped and inserted activities and measures performance metrics accordingly. In the implemented ProM plug-in, we assume that all possible unknown lifecycle type events that existed in the log are filtered out in advance and there are no composite tasks. Note that this is not a limitation of the performance analysis technique that we propose, as composite tasks can be unfolded and unrecognizable lifecycle types are ignored.

While providing a solid foundation, special mapping patterns for constructing Flexible models from the YAWL language and an extension to handle cancellation regions needed to be introduced to handle the added complexities of the YAWL language before the replay technique can be applied. Unlike Flexible models, YAWL models may contain tasks and conditions, but, unlike Petri nets, it is not mandatory to place conditions between tasks, which required special consideration for visualization. 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 replacer provides insight into the various cancellation actions carried out during process execution.

What makes the YAWL process log replacer unique is its support for complex constructs like multiple instances, cancellation regions, and OR-joins. The replacer in this work is based on the replacer for Flexible models (Adriansyah et al. 2010) and the replacer for Petri Nets (Adriansyah et al. 2011), which are shown to be superior against other replay techniques in handling OR splits/joins (Adriansyah et al. 2010). We refer to the work of Adriansyah et al. (2011) for theoretical proofs on the optimality of the replay result with respect to given cost of deviations. With a memory-efficient data structure to store state spaces based on the one used in Westergaard and Maggi (2011) backed with a hashmap for fast-lookup on the storage, performance of a replacer for Petri nets based on the A* algorithm in Adriansyah et al. (2011) is able to efficiently handle real-world cases. For instance, a real-world log of a building permit process from a municipality in the Netherlands that contains 103 unique cases with average number of twenty-five events per case (maximum forty-two events in a single case) can be replayed on a Petri net with thirty-one transitions and thirty-one transitions in just eighteen seconds. In total, 3,158,137 states were explored for the whole log and only 53MB of memory was consumed to store both the states and the traces. As the replay approach described in this paper slightly extends the one for Petri nets, we can expect similar performance results.
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. However, this replayer does not 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 colored 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. (2010, 2011) and van Dongen et al. (2009) and is an extension of the work on the YAWL replayer presented in Piessens et al. (2010). In the work by Adriansyah et al. (2010), a replay technique using Flexible models was proposed. These models do not have executable semantics, but their semantics are known only if a case was finished and logged. Therefore, this approach is robust to different control-flow constructs and, hence, applicable to processes with complex constructs. In Piessens et al. (2010), extensions were made in order to cope with all control flow constructs (e.g., cancellation, multiple instances, and OR-splits and joins) that YAWL supports. 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. However, in this paper we extended the approach further by removing the assumption that the logs used for performance analysis are taken directly from the YAWL system; hence, there is no guarantee or requirement of a 100 percent correspondence between the model and the log.

We proposed advanced replay techniques and provided updated software plug-in to support this extension. The given log and model are first compared to identify how a case in the log deviated from the given model, i.e., which activities were executed while they shouldn’t have and which activities were not executed while they should have. Once a case in the log is correctly mapped onto the model, the existing projection of performance information is used. As a result, performance analysis results become more widely applicable, especially for process improvement activities as existing logs from different systems can be compared against not only AS-IS YAWL models, but also with TO-BE models.

CONCLUSION

The paper proposes an approach to determining the performance metrics of either an AS-IS or a TO-BE business process specification given an event log of a corresponding AS-IS process. 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 visualized 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. The replayer uses a high-level language called Flexible models with cancellation to express the many complex control flow constructs supported by the YAWL language. The replayer is robust to deviations between the log and the model, and performance characteristics can be presented, even if the log and the model do not match fully. It is important, however, to keep in mind that for those cases where the log does not fully match the model, the resulting performance characteristics are based on assumptions about missing and inserted events. The quality of performance measurement is linear to the fitness value between the log and the model.

Obtaining the best replay match comes with a price of computational cost. To deal with complex processes and logs, a memory-efficient implementation of state space storage and lookup is crucial. In future work, the replayer’s 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 tasks are 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, sophisticated graph animations based on the resulting performance metrics could be developed.

REFERENCES


APPENDIX A

In this appendix, we show how activities for which multiple events have been recorded in the log are translated to nodes in a Flexible model. We explicitly chose to allow for a small set of fixed activity lifecycles and, using information in an event log, we select the most appropriate one based on available event types. We generally distinguish four event types referring to the start of an activity, the completion of that activity, and potentially the suspension or resumption of the activity. Cancellation is not explicitly modeled, as this is handled separately in the replayer. The dangling arrows indicate the first and last event, i.e. the points between which processing times are measured.

Figure 13: Translation of logged lifecycle events to Flexible model nodes.
APPENDIX B

In this appendix, we show how different YAWL constructs are mapped to nodes in a Flexible model. For activities, we show how the different join and split types are translated; for the multiple instances construct, we show both cases in which one, or more event types are present. Other combinations are logical extensions of the depicted translations.

Figure 14: Translation of YAWL constructs to Flexible model nodes.
ABOUT THE AUTHORS

Arya Adriansyah is a Ph.D. candidate in the Architecture of Information Systems group at Eindhoven University of Technology. He obtained his M.Sc. in Computer Science (2009) at the same university. His research interests include business process conformance checking, performance analysis, and visualization of research results.

Boudewijn van Dongen is an assistant professor in the Information Systems group of the Department of Mathematics and Computer Science of Eindhoven University of Technology, Eindhoven, The Netherlands. He received his Ph.D. in 2007, after successfully defending his thesis, entitled “Process Mining and Verification.” Currently, his research interests extend from process mining and process verification to supporting flexible processes and visualization of research results. Furthermore, he plays an important role in the development of the open-source process mining framework ProM, freely available from www.processmining.org.

David Piessens is a graduate student from Eindhoven University of Technology. He obtained his M.Sc. in Business Information Systems (2011) at the same university, specializing in Business Process Management. Presently he is pursuing a career as an SAP professional, focusing on Business Process Improvement and Business Intelligence.

Moe Thandar Wynn is a senior lecturer within the BPM group at the Queensland University of Technology in Brisbane, Australia. She received her Ph.D. in 2007, with the title “Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins.” Her current research interests include risk-aware business process management, cost-aware business process management, process mining, process verification, and process automation.

Michael Adams is a senior lecturer within the BPM group at the Queensland University of Technology in Brisbane, Australia. His research interests include process flexibility and exception handling, risk and cost awareness for processes, and Activity Theory. He is currently directly responsible for the ongoing development and maintenance of the YAWL system.
**Editors-in-Chief**

Marcus Rothenberger  
University of Nevada -- Las Vegas  

Mark Srite  
University of Wisconsin – Milwaukee  

Tuure Tuunanen  
University of Oulu

---

**Governing Board**

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kalle Lytinen</td>
<td>Case Western Reserve University</td>
<td>Las Vegas</td>
</tr>
<tr>
<td>Rajiv Kishore</td>
<td>State Univ of New York, Buffalo</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Douglas Vogel</td>
<td>City University of Hong Kong</td>
<td>Hong Kong</td>
</tr>
<tr>
<td>Ken Peffers</td>
<td>Emeritus &amp; Founding EIC</td>
<td>Las Vegas</td>
</tr>
<tr>
<td>Lars Mathiasson</td>
<td></td>
<td>Georgia State</td>
</tr>
</tbody>
</table>

**Senior Advisory Board**

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tung Bui</td>
<td>University of Hawaii</td>
<td>Hawaii</td>
</tr>
<tr>
<td>Brian L. Dos Santos</td>
<td>University of Louisville</td>
<td>Louisville</td>
</tr>
<tr>
<td>Robert Kauffman</td>
<td>Arizona State University</td>
<td>Las Vegas</td>
</tr>
<tr>
<td>Ken Kendall</td>
<td>Rutgers University</td>
<td>New Brunswick</td>
</tr>
<tr>
<td>Ephraim McLean</td>
<td>Georgia State University</td>
<td>Georgia</td>
</tr>
<tr>
<td>Edward A. Stohr</td>
<td>Stevens Institute of Technology</td>
<td>Georgia</td>
</tr>
</tbody>
</table>

**Senior Editors**

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roman Beck</td>
<td>University of Frankfurt</td>
<td>Buenos Aires</td>
</tr>
<tr>
<td>Kevin Crowston</td>
<td>Syracuse University</td>
<td>Michigan</td>
</tr>
<tr>
<td>Karlheinz Kautz</td>
<td>Copenhagen Business School</td>
<td>Copenhagen</td>
</tr>
<tr>
<td>Peter Axel Nielsen</td>
<td>Aalborg University</td>
<td>Aalborg</td>
</tr>
<tr>
<td>Jan Recker</td>
<td>Queensland Univ of Technology</td>
<td>Queensland</td>
</tr>
<tr>
<td>Jason Thatcher</td>
<td>Clemson University</td>
<td>Clemson</td>
</tr>
<tr>
<td>Gurpreet Dhillon</td>
<td>Virginia Commonwealth Univ</td>
<td>Virginia</td>
</tr>
<tr>
<td>Sirkka Jarvenpaa</td>
<td>University of Texas at Austin</td>
<td>Austin</td>
</tr>
<tr>
<td>Julie Kendall</td>
<td>Rutgers University</td>
<td>New Brunswick</td>
</tr>
<tr>
<td>Ying-Peng Liang</td>
<td>Nat Sun Yat-sen U, Kaohsiung</td>
<td>Kaohsiung</td>
</tr>
<tr>
<td>Timo Sarinen</td>
<td>Helsinki School of Economics</td>
<td>Helsinki</td>
</tr>
<tr>
<td>J. Christopher Westland</td>
<td>HKUST</td>
<td>HKUST</td>
</tr>
</tbody>
</table>

**Editorial Review Board**

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Murugan Anandarajan</td>
<td>Drexel University</td>
<td>Texas Pan American</td>
</tr>
<tr>
<td>Patrick Chau</td>
<td>The University of Hong Kong</td>
<td>Deakin University</td>
</tr>
<tr>
<td>Khalil Drira</td>
<td>LAAS-CNRS, Toulouse</td>
<td>MichiganDearborn</td>
</tr>
<tr>
<td>Peter Green</td>
<td>University of Queensland</td>
<td>Southern Mississippi</td>
</tr>
<tr>
<td>Peter Kueng</td>
<td>Credit Suisse, Zurich</td>
<td>United Arab Emirates</td>
</tr>
<tr>
<td>David Yuh Foong Law</td>
<td>National Univ of Singapore</td>
<td>Texas at Dallas</td>
</tr>
<tr>
<td>Vijay Mookerjee</td>
<td>University of Texas at Dallas</td>
<td>Texas at Dallas</td>
</tr>
<tr>
<td>Georg Peters</td>
<td>Munich Univ of Appl. Sciences</td>
<td>Dallas</td>
</tr>
<tr>
<td>Rahul Singh</td>
<td>Univ of N. Carolina, Greensboro</td>
<td>Syracuse</td>
</tr>
<tr>
<td>Issa Traore</td>
<td>University of Victoria, BC</td>
<td>Indiana</td>
</tr>
<tr>
<td>Jonathan D. Wareham</td>
<td>Georgia State University</td>
<td>BC</td>
</tr>
</tbody>
</table>

**JITTA is a Publication of the Association for Information Systems**  
ISSN: 1532-3416