BP-Mon: Query-Based Monitoring of BPEL Business Processes *

Catriel Beeri
Hebrew University
cbeeri@cs.huji.ac.il

Anat Eyal
Tel Aviv University
anate@post.tau.ac.il

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Alon Pilberg
Tel Aviv University
allonpil@post.tau.ac.il

1. INTRODUCTION

A Business Process (BP for short) consists of some business activities undertaken by one or more organizations in pursuit of some particular goal. It often interacts with other BPs of the same or other organizations and the software implementing it is rather complex. Two complementary instruments facilitate the design, development, and management of this complex software. The first is the use of *standards*. In particular, the recent BPEL standard (Business Process Execution Language [5]) provides an XML-based language to describe the operational logic and execution flow of the BP, as well as the interfaces it exposes to other BPs. A BP specification written in BPEL can be automatically compiled into an actual code that implements the BP, and can be executed on a BPEL server. The second instrument is the use of *supporting BP management tools* for (1) designing the BP BPEL specifications, (2) analyzing the design, (3) monitoring the BPs at run time, and (4) analyzing, posteriorly, the process execution traces (logs). Together they provide an essential infrastructure for companies to design business processes, optimize them, reduce operational costs, and ultimately increase competitiveness.

The BP-Mon system described in this paper is part of `BP-Suite`, a novel tool suite based on the BPEL standard. `BP-Suite` offers a uniform, query-based, user-friendly interface that gracefully combines the analysis of process specifications, monitoring of run time behavior, and log analysis, for a comprehensive process management. `BP-Suite` consists of three tightly coupled query subsystems: `BP-QL` allows one to query and analyze BP specifications; `BP-Mon` allows for monitoring the execution of BP instances; and `BP-Ex` allows for a posteriori analysis of the their execution traces (logs). The three sub-systems are all based on *the same* simple, intuitive, graphical query language, whose GUI is very similar to that used by commercial vendors for the *design* of BPEL processes. This is an important feature of the system, as it allows (a) fast learning of the language and (b) simultaneous formulation, by a BP designer, of both the BP specification and the verifcation/monitoring/log analysis queries over it.

As a simple example of the different types of BP analysis, consider a BP of a Web-based auctioning business. An analysis of the BP specification (hence of the potential run-time behavior of the BP), may allow the manager to assure that certain security policies are enforced. For instance, she may want to query the specification to assure that in no place a buyer can place a bid without giving her credit card details first. Similarly, run-time monitoring of process execution may allow the manager to detect fraud attempts and track services usage and performance. Finally, querying and analyzing, posteriorly, the process execution traces (logs) may allow the manager to identify usage trends and optimize the process accordingly. Observe that querying of the potential behavior of BPs and monitoring/analysis of the actual run-time behavior are complementary. Queries on the specification can be used to focus on (the parts of) the BPs that require monitoring/log analysis. Conversely, run-time monitoring/log analysis can be used for analysis of process properties that cannot be statically determined by querying the specification.

Contributions. In this paper, we present a short overview of one sub-system of `BP-Suite` – `BP-Mon` , which allows one to monitor process instances at run-time. Concentrating on this subsystem, we will demonstrate the flexibility and power of the `BP-Suite` query language. As mentioned above, essentially *the same query*, can be used, under different interpretations, to analyze a BP specification, monitor execution of its instances at run time, and analyze the resulting logs. `BP-Mon` allows users to visually define monitoring tasks and associated reports, using a simple intuitive interface similar to those used for designing BPEL processes. We will present here the language features as well as its implementation, for the context of BP monitoring. An interesting characteristic of the implementation is that BP-Mon queries are translated to BPEL processes that run on the same execution engine as the monitored processes. Our experiments (see [3]) indicate that this approach incurs very minimal overhead, hence is a practical and efficient approach to monitoring.

The paper is organized as follows. Section 2 provides some background and describes the main challenges in BPEL BPs monitoring. Section 3 then gives an overview of the `BP-Mon` system, describes its implementation, and explains how it addresses these challenges. Finally we conclude in Section 4 by a brief overview of the distinct challenges encountered when developing the other system components - `BP-QL` and `BP-Ex` - implementing the same query language for the different contexts of BP analysis and log analysis.

2. BACKGROUND AND CHALLENGES

We start with some background on current BP Management Systems and the challenges in monitoring BPs.

As mentioned above, many enterprises nowadays use business processes, based on the BPEL standard, to achieve their goals. Since the BPEL syntax is quite complex, commercial vendors offer systems that allow to design BPEL process specifications via a visual interface, using an intuitive view of the process, as a graph

*The research has been partially supported by the European Project EDOS and the Israel Science Foundation.

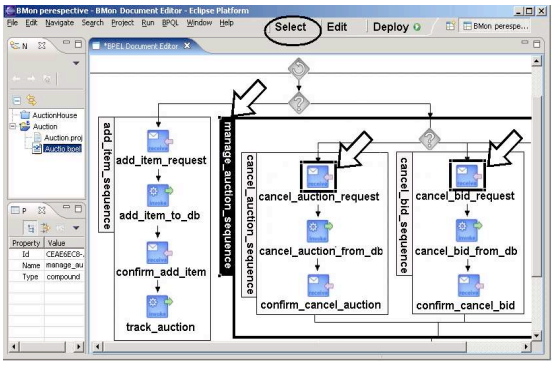

Figure 1: Selection from a BPEL specification.

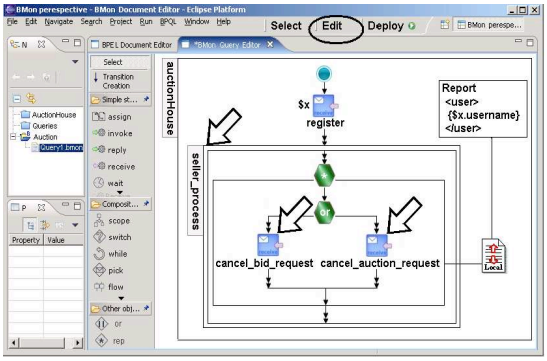

Figure 2: Edit query.

of activity nodes connected by control flow edges. An example of such design is depicted in Figure 1 that shows (part of) the BPEL specification of an auctioning BP (ignore the thick edges for now). These designs are automatically converted to their XML representation, then automatically compiled into executable code that implements the described BP.

An *instance* of a BP specification is an actual running process (that follows the logic described in the specification), that includes specific decisions, real actions, and actual data. BP Management Systems allow to trace process instances - the activities they perform, messages sent or received by each activity, values of variables used by the process, performance metrics - and send this information as events in XML format to a *monitoring* system (often called BAM – Business Activity Monitoring – system).

Monitoring the execution of such processes for interesting patterns is critical for enforcing business policies and meeting efficiency and reliability goals. For some intuition about the type of monitoring that a given BP may require, let us consider again the above mentioned manager of a Web-based auctioning business. Monitoring of process executions may allow the manager, among others, to guarantee fair play, detect frauds, and track services usage and performance. She can ask, for instance, to be notified whenever an auctioneer cancels bids too often, or when buyers attempt to confirm bids without first giving their credit details, so that she can block their actions. Similarly, being notified whenever the average response time of the database in a given service passes a certain threshold allows her to fix the problem or switch to a backup database. In general, monitoring encompasses the tracking of particular patterns in the executions of individual processes or in the interaction between different processes, as well as the provision of statistics on the performance of some processes or the system.

Typical monitoring systems (e.g. [1, 13, 15, 14]) are composed of three layers: one that absorbs the stream of events coming from the BP execution engine; another that processes and filters events, selects relevant event data and automatically triggers actions; and a dashboard that allows users to follow the processes progress, view custom reports and statistics on the processes and send alerts.

Although rather powerful, most BAM systems were developed for proprietary enterprise workflow management and address the needs of such systems. But the dynamic open nature of modern BPs pose new requirements, demanding, on the one hand, tighter surveillance, and on the other hand, a lighter, more add hoc, deployment:

Execution patterns. When monitoring BP instances, users may be interested in identifying certain execution patterns in a process flow (e.g. a buyer that attempts to confirm bids without first giving her credit details), as well as in retrieving the relevant parts of the flow (e.g. the actions sequence that the buyer followed, after registering, to bypass the request for credit card). Existing monitoring tools [15, 7] allow users to filter individual events based on their type and data values only, but do not consider the flow.

Flexible granularity. The execution of a BP instance may be abstractly viewed as a nested set of DAGs (Directed Acyclic Graphs) The DAGs structure captures the execution flow of the instance; the nesting is due to the fact that processes contain composite activities, each with a complex internal execution flow (itself represented by a DAG). When monitoring a process, users may wish to consider certain activities as black boxes, but zoom-in, possibly recursively, into some other activities. Thus, there is a need to provide users the flexibility to monitor processes at varying levels of granularity. This is extremely difficult, if not impossible, in most existing tools: selecting the relevant entries from all the possible events, without being able to reference the process flow, is complex and requires intimate knowledge of the monitored application.

Easy deployment. As mentioned above, the BPEL standard facilitates the design, development and deployment of BPs: BPs are specified in a high level manner and the specifications are automatically compiled into executable code that can, in principle, run on any BPEL application server [18]. Analogously, it is desirable that a monitoring task would be defined in high-level manner, and be compiled, and easily deployed, on whatever BPEL application server chosen for the monitored BP. In existing monitoring tools, however, the selection rules for events are written in proprietary, non-poratble, format. Furthermore, their definition is not trivial and is typically done by the system administrator when a new system is deployed, or when business requirements change.

3. `BP-Mon`

The `BP-Mon` (Business processes Monitoring) monitoring system addresses these issues. We next describe the system and its implementation.

3.1 The `BP-Mon` solution

`BP-Mon` system makes the following contributions.

Query language. The system is based on an intuitive graphical query language that allows for simple description of the execution patterns to be monitored. A tight analogy between the graphical interface used by commercial vendors for the *specification* of BPs and our graphical query interface allows intuitive design of moni-

toring tasks.

The BP-Mon query language is an adaptation of the sister query languages used in BP-QL ([2]) (for specification analysis) and BP-Ex (for logs analysis), to run-time monitoring. In all three query languages, the data (i.e. the BP specification or its execution traces) is abstractly viewed as a nested set of DAGs (Directed Acyclic Graphs). A query consists of two parts. The first (which is identical across all the three sub-systems) specifies the execution patterns that are of interest to the user. The execution patterns used here extend string regular expressions to (nested) process DAGs. They can describe sequential and parallel execution of activities, possibly with repetitions and/or alternatives, and allow the user to zoom in inside compound BP activities or view them as black boxes. As an example, the execution pattern in the BP-Mon query depicted in Figure 2 (ignore the thick edges for now), searches for users registered as sellers, that repetitively cancel bids or cancel auctions. The second part of the query, (which is specific to BP-Mon) consists of *Report icons* that can be attached to the patterns. In our example, the report icon appears on the right side of Figure 2, with the report format defined in the top right box. Every time the activity node attached to this report icon is matched, the system will issue a report. These reports allow to notify users of occurrences of the monitored patterns, report relevant data (including relevant execution paths), and possibly invoke corrective actions.

Deployment. To support flexible deployment, our system compiles a BP-Mon query q into a BPEL process specification S, whose instances perform the monitoring task. As for all standard BPEL specifications, S can now be automatically compiled into executable code to be run on the same BPEL application server as the monitored BP. Our experiments [3] prove that the resulting monitoring is extremely efficient and incurs only very minimal overhead.

Query evaluation and optimization. Users should be notified as soon as their patterns of interest occur. BP-Mon uses an efficient automata-based algorithm that finds the *first match* of a query (execution pattern) in a stream of events of a given process instance. A novel optimization technique that prunes redundant monitoring activities based on an analysis of the process BPEL specification, speeds up computation, by focusing on the relevant parts of the trace, and filtering out events which are irrelevant or inconsistent with the query.

Discussion. We have mentioned above that events are sent to monitoring systems in standard XML format. A natural question is why not use XQuery, coupled with some XML stream-processing engine [11, 9, 19, 16], to process this stream of events? A key observation is that the XML elements in this stream describe individual events. To express any non-trivial query about a process execution flow, one needs to write a fairly complex XQuery query, that performs an excessive number of joins, and can hardly (if at all) be handled by existing streaming engines[8, 17, 6].

Furthermore, standard XML stream processing would still be inadequate for the task, even if a more query-friendly nested XML representation, that reflects the flow, had been chosen for the data: XML stream engines manage tree-shaped data and not DAGs. More importantly, they expect to receive the tree elements in document order and process siblings sequentially, as they arrive [8, 12, 10]. But the events flow in BPs does not necessarily follow this order since parallel activities interleave. Here, parallel processing, that processes each event according to its position in the flow is called for; this is provided by BP-Mon.

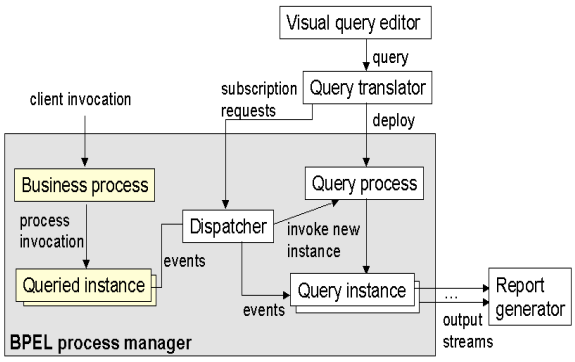

Figure 3: Architecture.

In summary, BP-Mon allows one to design complex monitoring tasks that deal with both events and flow; it offers easy, user-friendly design of such tasks; and it compiles these tasks into standard BPEL processes, thus providing easy deployment, portability, and minimal overhead.

3.2 The BP-Mon Implementation

A first demo of the BP-Mon system was given in SIGMOD'07 [4]. A full description of the implementation and the used optimization techniques is given in [3]. We briefly overview here the main system components. The BP-Mon system runs on Windows XP Professional, JBoss AS 4.0.4. Oracle BPEL Process Manager 10.1.2. with Oracle 9i database. The system architecture is depicted in Figure 3.

Visual editor. BP-Mon queries are written via a visual editor, in one of two modes. The user can draw the query patterns that she wishes to monitor from scratch, using a drag-and-drop items palette. Or, starting from a BPEL specification of a BP p, use a wizard to create a query to monitor p, as follows: The user marks the nodes of the specification that she wishes to include in the query. Then, by one click, a query(pattern) draft is created, where non selected nodes are omitted and the selected nodes are connected with special edges that reflect their flow and zoom-in relationship in the specification. The user can then add conditions on the node values, detail the report data she wishes to see, make some final adjustments, and click a button to deploy the query on a BPEL server.

As an example, Figure 1 shows (part of) the BPEL specification of the auctioning BP, with the thick edges pointing at the nodes selected (in this part) by the user. The generated monitoring query, after some user adjustments, is shown in Figure 2, ready to be deployed. The query monitors auctioneers for repeated cancelations of bids or auctions. We do not discuss the syntax here, for space constraints (for a detailed description see [3]), but only point out that the query indeed looks very much like the specification. This uniform intuitive graphical interface allows for natural query formulation. The visual interface is implemented as an Eclipse plug-in, similarly to Oracle BPEL designer; both products can run simultaneously in the same framework.

Query translator. As mentioned in Section 3, to support flexible deployment, the system compiles BP-Mon queries into BPEL specifications. This is done by the *Query Translator* module, that basically translates each activity into a state, implemented as a compound activity consisting of two components, one in charge of reading the incoming events, the other in charge of event processing and backtracking. The specification $S(p)$ generated for a

query pattern p describes a process (essentially a sophisticated automaton) that will perform the monitoring task for p. $S(p)$ is called the *Query Process* (QP for short) of p. The QP is deployed onto the BPEL server where the instances of p are executed. Several QPs, monitoring the same or different processes, may be deployed on a server. Note that in principle one may even have queries that monitor the execution of other queries!

Dispatcher. For each query, our system generates one QP instance per monitored BP instance. Processes and instances in BPEL servers have ids, and these are used by the dispatcher module to dispatch the BP instances events to the right QP instances. The dispatcher module subscribes to relevant events of the queried BPs when a query is deployed, and receives the relevant events generated by instances of these BPs (as described in Section 2). The first event from a new BP instance causes the dispatcher to create a new instance of relevant QPs. Further events are delegated to the running QP instances.

Report generation. Finally, a successful matching for the query pattern triggers the generation of a corresponding report or corrective action. Two reporting modes are available: *local*, where an individual report is issued for each process instance, and *global*, that spans all the BP instances. For each report one can specify when it should be issued (e.g. the first time that the pattern occurs, at periodic time intervals, or when certain conditions are satisfied) and what should be the structure of the output (in XML format) or the actions triggered at this point. Reports may include sliding window aggregations like average, max, min, count, sum. The different types of reports are defined using the system's graphical editor (see above) and can be attached to various parts of the query patterns.

The implementation of queries by translation into BPEL processes, then running them on the same server as the queried processes, has two main advantages: Portability of queries between BPEL engines; and a great simplification of the software devolvement. The implementation exploits the infrastructure provided by such engines for parallel and distributed process management, and software composition. The price payed for this is the extra load on the BP server who now needs to also run query instances. To estimate the overhead incurred by running the query on the same server, the performance impact on the queried processes, the scalability of the solution, and the effectiveness of the optimizations, we conducted an experimental comparative study of the BPEL server performance with and without our monitoring processes. The results, reported in [3], show that the resulting monitoring is extremely efficient and incurs only minimal overhead.

4. QUERYING BP SPECIFICATIONS AND EXECUTION LOGS

As mentioned above, `BP-Mon` is part of the `BP-Suite` system that allows users to use the same query language also for analyzing BP specifications and execution logs. While uniform from the user's perspective, the underlying implementation of the three subsystems differs greatly, addressing distinct challenges that arise from their particular context.

In the `BP-QL` sub-system [2], the same graphical query language is used for querying BP *specifications*. There, the goal is to be able to retrieve specifications with certain properties (e.g. where an execution path of a particular pattern is (not) possible). The query evaluation algorithm there relies on modeling specifications and queries as graph grammars. Query evaluation amounts, intuitively, to computing the intersection between the corresponding grammars.

In the `BP-Ex` sub-system [20], the query language is used to analyze, posteriorly, the process execution traces (logs). To evaluate a query, sub-traces of the shape specified by the query pattern need to be retrieved and analyzed. A key challenge in this context is the large amount of data that need to be processed. The solution here is based on mapping the graphical queries to a dedicated algebra, with special algebraic rewrite rules that allow to optimize performance.

5. CONCLUSTION

This paper presents one sub-system of `BP-Suite` – `BP-Mon` , a novel query language for monitoring BPs. `BP-Mon` offers a high level intuitive design of monitoring tasks. A novel optimization technique exploits available knowledge on the BP structure to speed up computation. Further optimization to be studied may include pattern simplifications, e.g. replacing non-transitive edges with transitive ones and reducing pattern nesting by eliminating unnecessary compound activities.

6. REFERENCES

[1] BEA. Bea aqualogic bpm suite. http://www.bea.com/bpm/.

[2] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying Business Processes. In *Proc. of VLDB*, pages 343–354, 2006.

[3] C. Beeri, A. Eyal, T. Milo, A. Pilberg. Monitoring business processes with queries. In *Proc. of VLDB*, pages 603–614, 2007.

[4] C. Beeri, A. Eyal, T. Milo, A. Pilberg. Query-based monitoring of BPEL business processes. In *SIGMOD*, pages 1122–1124, 2007.

[5] Business Process Execution Language for Web Services, 2003. http://www.ibm.com/developerworks/library/ws-bpel/.

[6] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring streams - a new class of data management applications. In *VLDB*, pages 215–226, 2002.

[7] M. Castellanos, F. Casati, M. Shan, and U. Dayal. ibom: A platform for intelligent business operation management. In *ICDE*, pages 1084–1095, 2005.

[8] Y. Diao and M. J. Franklin. Query processing for high-volume xml message brokering. In *VLDB*, pages 261–272, 2003.

[9] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, 2005.

[10] C. Y. Chan, P. Felber, M. N. Garofalakis, R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *ICDE*, 2002.

[11] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[12] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing xml streams with deterministic automata. In *ICDT*, pages 173–189, 2003.

[13] HP. Openview bpi. http://www.hp.com.

[14] IBM. WebSphere Business Monitor. http://www-304.ibm.com/jct03001c/software/integration/wbimonitor.

[15] Ilog jviews. http://www.ilog.com/products/jviews/.

[16] N. Koudas and D. Srivastava. Data stream query processing. In *ICDE*, 2005.

[17] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003.

[18] Oracle BPEL Process Manager 2.0 Quick Start Tutorial. http://www.oracle.com/technology/products/ias/bpel/index.html.

[19] F. Peng and S. S. Chawathe. Xpath queries on streaming data. In *SIGMOD*, pages 431–442, 2003.

[20] T. Sterenzy. BP-Ex: Optimized Analysis of Business Processes Logs. M.Sc Thesis, Tel Aviv University, 2008.