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 a group of business activities undertaken by one or more organizations in pursuit of some particular goal. It often interacts with other BPs carried by the same or other organizations, and the software implementing it is fairly complex. Standards facilitate the design, deployment, and execution of BPs. In particular, the recent BPEL standard (Business Process Execution Language [3]), provides an XML-based language to describe the interface between the participants in a process, as well as the full operational logic of the process and its execution flow. BPEL specifications are automatically compiled into executable code that implements the described BP and runs on a BPEL application server.

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, consider a manager of a Web-accessible 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. This provides an essential infrastructure for companies to optimize business processes, reduce operational costs, and ultimately increase competitiveness.

The goal of this demonstration is to present `BMon` (Business process Monitoring), a novel query language and system for monitoring business processes. `BMon` 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 illustrate the language features as well as its implementation. An interesting characteristic of the implementation is that `BMon` queries are translated to BPEL processes that run on the same execution engine as the monitored processes. Our experiments indicate that this approach incurs very minimal overhead, hence is a practical and efficient approach to monitoring.

2. BACKGROUND

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 specifications via a visual interface, using an intuitive view of the process, as a graph of activity nodes connected by control flow edges. Designs are automatically converted to BPEL specifications, which are 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), which 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, performance metrics - and send this information as events in XML format to a *monitoring* systems (often called BAM – Business Activity Monitoring – systems). Typical monitoring systems (e.g. [1, 10, 11]) 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 events 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 enterprise workflow management and address the needs of such companies. 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 [11, 4] 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 with 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 facilitate 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 [13]. Analogously, it is desirable that a monitoring task would be defined in a declarative 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 format and are not portable. 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. THE BMon SYSTEM

The BMon (Business processes Monitoring) system presented in this demonstration addresses these issues. It 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 BP and our graphical query interface allows intuitive design of monitoring tasks. The execution patterns in BMon extend string regular expressions to (nested) DAGs. They can describe sequential and parallel execution of activities, possibly with repetitions and/or alternatives, and allow to indicate the granularity levels to be employed for different components of a monitoring task. The BMon reporting facility allows to notify users of occurrences of the monitored patters, report relevant data (including relevant execution paths), and possibly invoke corrective actions.

Implementation To support flexible deployment, our system compiles a BMon 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 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. BMon uses an efficient automata-based algorithm that finds the *first match* of a query (execution pattern) in a given process trace. A novel optimization technique that prunes redundant monitoring activities based on an analysis of the process BPEL specification, allows to speed up computation, by focusing on the relevant parts of the trace.

In summary, BMon allows 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.

Related work and discussion. In a previous work [2] we proposed to use a graphical query language for querying BP *specifications*. There, the goal was to be able to retrieve specifications with certain properties (e.g. that an execution path from activity A to activity B is possible), and the solution relied on modeling specifications and queries as graph grammars. In contrast, our work here is concerned with querying the *actual execution* of process instances (e.g. to find when an execution path that started at activity A arrives to activity B), and the solution is based on automata construction. Indeed the two works are complementary: The query language of [2] can be used to focus on parts of the PBs that require monitoring, while monitoring can be used to to check at runtime properties that cannot be statically determined by querying the specification.

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 [8, 6, 14, 12], 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[5, 7]. 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[5, 9]. 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 BMon .

4. DEMONSTRATION

To illustrate the features of BMon we will consider a set of BPEL business processes used by a Web-accessible auctioning business. The processes include seller and buyer services as well as accounting, credit, and price comparison facilities. They reside and operate on three peers (three computers, in our demo). We will show how the intuitive graphical BMon query interface is used to define monitoring queries for a variety of critical tasks such as fraud detection, SLA (service level agreement) maintenance, and general business management. The ease of query formulation will be illustrated by comparing our graphical query interface to that used by commercial vendors for the specification of business processes (e.g. [13]); there is a tight analogy between how processes are specified and how they are monitored.

The 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 Fig. 1. The demonstration will illustrate each of the components and their interaction. The visual interface is implemented as an Eclipse plug-in, similarly to Oracle BPEL designer; both products can run simultaneously in the same framework.

Visual editor. BMon 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 adjustment, and

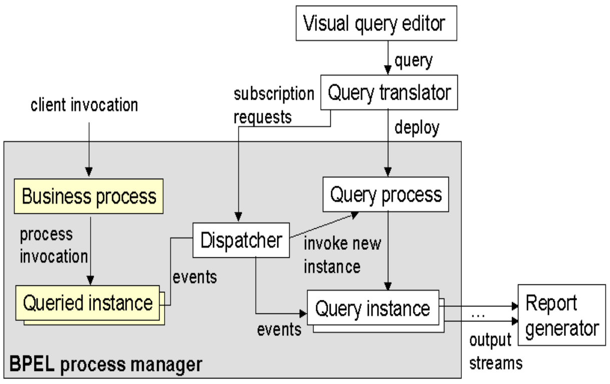

Figure 1: Architecture.

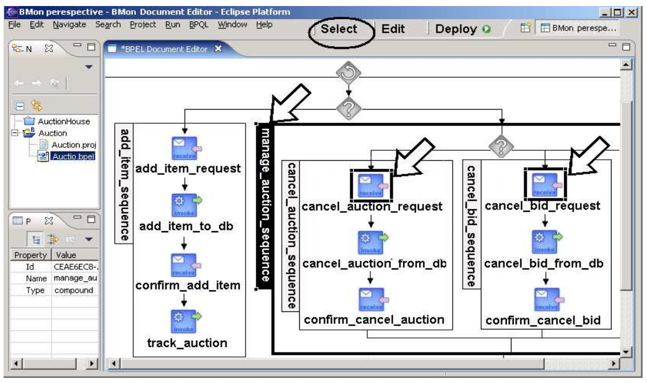

Figure 2: Selection from a BPEL specification.

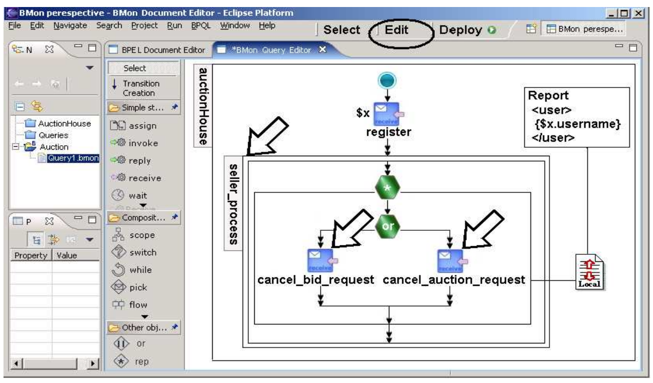

Figure 3: Edit query.

click a button to deploy the query on a BPEL server.

We will illustrate in the demonstration each of these two modes and show how, starting from the BPEL specification of the auctioning BP, one can easily formulate monitoring queries for it. As an example, Figure 2 shows (part of) the BPEL specification of the auctioning BP with the nodes selected (in this part) by the user. The generated monitoring query, after some user adjustments, is shown in Figure 3, ready to be deployed. We do not discuss the syntax here, but will naturally detail it in the demonstration.

Query translator. As mentioned in Section 3, to support flexible deployment, the system compiles BMon queries into BPEL specifications. This is done by the *Query Translator* module. 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! In the demonstration we will show the BPEL specification generated for the defined queries and the deployment of the corresponding QP. We will demonstrate how these monitoring tasks can be themselves monitored, e.g. to follow their progress.

Dispatcher. Now that the QPs are deployed onto the BPEL server, we will demonstrate what happens at runtime. The *dispatcher* module is responsible for the run-time mapping between the events of BP instances and the QPs. It 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. A successful matching for the query pattern associated with a report 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 should it be issued (e.g. at the first time that the pattern occurs, at periodic time interval, 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. We will show how different types of reports can be defined using the system's graphical editor (see above) and be attached to various parts of the query patterns, to be reported on match for the corresponding part.

To conclude the demonstration, we will briefly present experimental results that demonstrate that the resulting monitoring is extremely efficient and incurs only very minimal overhead for the running PBs.

5. 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 *VLDB*, 2006.

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

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

[5] Y. Diao and M. J. Franklin. Query processing for high-volume xml message brokering. In *VLDB*, 2003.

[6] D. J. Abadi et al. The design of the borealis stream processing engine. In *CIDR*, 2005.

[7] R. Motwani et al. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003.

[8] S. Chandrasekaran et al. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

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

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

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

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

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

[14] F. Peng and S. S. Chawathe. Xpath queries on streaming data. In *SIGMOD*, 2003.