BP-Ex – A uniform query engine for Business Process Execution traces *

Eran Balan Tova Milo Tal Sterenzy
Tel Aviv University
{eranbala,milo,sterenzy}@post.tau.ac.il

Categories and Subject Descriptors
H.4 Information Systems [**Information Systems Applications**]: H.4.1 Office Automation—*Workflow management*

ABSTRACT
Many enterprises nowadays use business processes, based on the BPEL standard, to achieve their goals. Analyzing the execution of such processes is critical for enforcing business policies and meeting efficiency and reliability goals.
The BP-Ex system presented in this demo is an important component of BP-Suite, a novel tools suite based on the BPEL standard, which offers a uniform, query-based, user-friendly interface for BP analysis. BP-suite allows to gracefully combine the analysis of process specifications, monitoring of run time behavior, and posteriorly querying of execution traces (logs), for a comprehensive process management. BP-Ex is the BP-Suite query engine for process execution traces. The goal of this demo is to highlight the particular challenges that had to be addressed to support the suite's uniform, intuitive query interface, over (possibly very large) execution traces, and to demonstrate the novel optimization techniques that had to be developed for that.

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 BPEL standard (Business Process Execution Language [4]) provides an XML-based language to describe the operational logic and execution flow of the BP, as well as the interface it exposes to other BPs. BP specifications can be written in BPEL in a uniform manner and then automatically compiled into an actual code that implements the BP and runs on a BPEL server. The second instrument is the use of *supporting BP management tools* for (1) designing the 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 optimize business processes, reduce operational costs, and ultimately increase competitive-ness [6].

As a simple example for the importance of (the different types of) BP analysis, consider a BP of a travel agency offering its Web users various travel-related services, such as flight, train and hotel reservation, as well as corresponding payment services. An analysis of the BP specification, (hence of the potential run-time behavior of the BP), may allow the manager to assure that certain company policies are enforced. For instance, she may want to query the specification to assure that in no place a customer can confirm a reservation 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 the querying of the potential behavior of BPs and the 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/logs analysis. Conversely, run-time monitoring/logs analysis can be used to complement the analysis of process properties that cannot be statically determined by querying the specification.

The BP-Suite system presented in this demo is based on the BPEL standard and offers a uniform, query-based, user-friendly suite of tools that allow to gracefully combine the analysis of process specifications, monitoring of run time behavior, and execution traces analysis, for a comprehensive process management. Specifically, the demo will focus on BP-Ex, the sub-system that supports querying of process execution traces.

Overall, BP-Suite consists of three tightly coupled query subsystems: BP-QL allows to query and analyze BP specifications. BP-Mon allows to monitor process instances at run-time. BP-Ex allows for a posteriori querying of the process execution traces. 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 (a) it allows faster learning curve of the language and (b) it allows simultaneous formulation, by the BP designer, of the BP specification and verifcation/monitoring/logs analysis queries over it.

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

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

The first two sub systems, BP-QL [2] and BP-Mon [3] were demonstrated in VLDB'06 and SIGMOD'07, resp. The present demonstration aims to complete the picture, presenting the third, last subsystem, BP-Ex. The goal here is twofold. First, looking at the system as a whole, we wish to demonstrate the flexibility and power of its query language, showing how essentially *the same query*, can be used, under different interpretations, to analyze the BP specification, monitor the BP at run time, and analyze the logs. Second, focusing on the new BP-Ex component, the goal is to highlight the particular challenges that had to be addressed to support such flexible querying of (possibly very large) execution traces, and to demonstrate the novel optimization techniques that had to be developed for that.

2. BACKGROUND

A key property of BP-Suite is the uniformity, across all BP management tasks, of the data model, query language, and user interface. To explain the novelty here, we start with some background on current BP Management Systems.

Given the importance of BP management and the large market size, many commercial vendors and research projects offer tools that address (some subset of) the above mentioned needs. To fully cover all management aspects, users often need to use several distinct tools. A key difficulty here, from a user perspective, is the diversity of models, abstraction levels, specification/query languages, and GUIs that these tools employ. To better understand this, let us briefly overview the main tasks involved in BP management and the families of tools typically available for each task.

BP specification. Many enterprizes nowadays use the BPEL standard to define their BPs. As mentioned above, BPEL is an XML-based language that allows to describe the BP operational logic as well as the interface it exposes to other BPs. 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. The designs are automatically converted to BPEL specifications. These are then automatically compiled into executable code that implements the described BP and runs on a BPEL application server [17].

Specifications analysis. There has been a vast amount of previous work in the general area of program analysis and verification (see e.g. [15] for a sample), and more specifically in the analysis of BPEL specifications [9, 7, 18]. These works mostly consider logic-based query languages where queries, formulated as logic formulas, test if the runs of the application or program satisfies a certain property; a witness counter example is provided if not. The query languages and their UI are typically quite different from that used for the BP specification and require different expertise.

Run-time monitoring. An *instance* of a BP specification is an actual running process which includes specific decisions, real actions, and actual data. BPEL servers allow to trace process instances - the activities they perform, messages sent or received by each activity, variable values, performance metrics - and send this information as events (in XML format) to monitoring systems (often called BAM - Business Activity Monitoring - systems), or log the data. (This log is called an *execution trace*).

Typical monitoring systems (e.g. [1, 13, 12]) allow users to specify

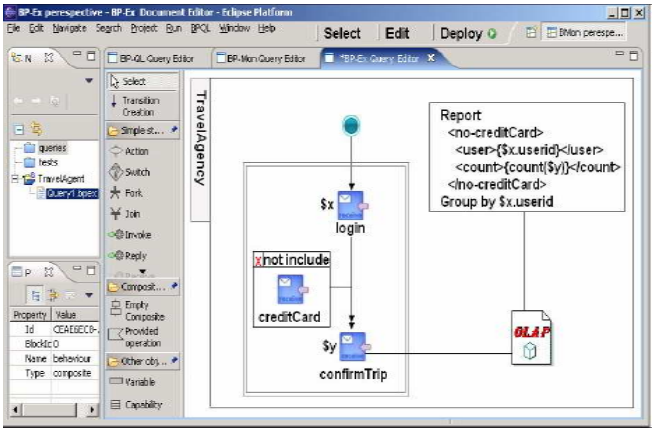

Figure 1: Query example

events of interest, and actions to be performed when the events are identified. Events may be atomic or composite (i.e. consist of a group of other atomic or composite event). Detection and processing of (composite) events has been an active research area since the early 90's. Rich event algebras have been proposed for describing composite events, and sophisticated evaluation and optimization techniques have been developed for their detection [16]. While a few systems use a high level BPEL-like GUI [12], others employ a much lower abstraction level, requiring intimate knowledge of both the monitored application and the specific events emitted by each activity.

Execution traces analysis. There is a large scope of research that deals with post-analysis of BP executions. The common approach to analyzing process traces is to gather and store them in a data warehouse and then apply various techniques for analysis and reporting [5, 6, 10]. Most BP management systems provide some reporting capabilities showing basic process statistics such as performance and status information. OLAP tools (drill down, roll up, slicing, dicing, etc.) are proposed to derive reports in different levels of abstractions, aggregations, and perspectives for analyzing, understanding, and optimizing processes (e.g. [8]). Other works use business intelligence, i.e. data mining techniques. Some examples are, BPI [5], a tool suite that allows to obtain explanations and predictions on process metrics and behaviors, and VisImpact [11] that uses correlation and classification analysis techniques to abstract important business impact factors.

Although rather powerful, this diversity of tools, models, and languages, makes the management of BPs rather intricate. The BP-Suite system presented in this demonstration addresses this problem by offering a uniform suite of tools, all based on a *standard BPEL graphical GUI* and using the *same data model and query language*. Together they allow an intuitive and comprehensive process analysis, including analysis of process specifications, monitoring of run time behavior, and execution traces analysis.

3. BP-EX OVERVIEW

We next provide a brief overview of the main component of the BP-Ex subsystem, the focus of this demonstration, highlighting the novelty and contributions.

It should be emphasized that our goal in the development of BP-Ex was not to compete with existing commercial tools for analyzing

execution traces. Rather, it was to prove that it is possible to base such a tool on the same data model and UI as used for BP design, and on the same query language as used by us for the analysis of BP specifications and their run-time monitoring. That is, to show that uniformity, expressibility, and good performance are *not* contradictory.

Query Language. The BP-Ex query language is an adaptation of the sister query languages used in BP-QL (for specification analysis) and BP-Mon (for run time monitoring), to the analysis of execution traces. In all three query languages, the data (i.e. the BP specification and its execution traces) is abstractly viewed as a nested set of DAGs (Directed Acyclic Graphs). The DAGs structure captures the execution flow of the process; the nesting is due to the fact that processes contain composite activities with complex internal execution flow (itself represented by a DAG). A query consists of two parts. The first (which is identical across all the three sub-system) 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 to zoom in inside compound BP activities or view them as black boxes. The second part of the query (which is specific to BP-Ex) consist of *OLAP icons* that can be attached to the patterns. These allow to compute aggregate functions, over the retrieved sub-traces, along different dimensions requested by the user, and generate corresponding reports.

Multi-Embedding. To evaluate a query, sub-traces of the shape specified by the query pattern need to be retrieved and analyzed. An occurrence of a given pattern in the execution trace is called an *embedding*. Identifying and handling each embedding individually may be rather expensive performance-wise. Our solution is based on the observation that an execution trace may contain many embeddings of the same pattern, and that embeddings of the same (or different) patterns may overlap. To exploit this we designed a novel compact data structure called Multi-Embedding (ME for sort) that "factorizes" multiple matches of the same trace activities and provides a concise representation for the existing embeddings. Intuitively, an ME can be viewed as a graph with shape similar to that of the query pattern and where (1) each ME node records all the trace nodes that were matched to the corresponding pattern node, and (2) each ME edge records the edge/path relationships between the trace node recorded by its endpoints. The ME can be computed efficiently and is used in query evaluation as explained below.

Algebra and logical optimization. To evaluate BP-Ex queries, the graphical query representation is mapped into an algebraic expression. The algebra consisted of a pattern-matching operator that retrieves the sub-traces matching the query pattern, as well as select, project, join, and standard OLAP-style operators for performing aggregation.

To allow for efficient processing, rather than applying naively the operators on the individual sub-traces, the algebraic query is translated into an equivalent ME-based expression that operates directly on the ME. In particular, each of the algebra operators has a corresponding ME variant. This includes a pattern-matching operator that builds the ME, select, project, and join operators that operate on it, as well as operators for performing OLAP-style analytical processing over the ME. Dedicated algebraic rewrite rules allow to optimize performance, by reordering operations and pushing selection conditions into the pattern matching (ME construction).

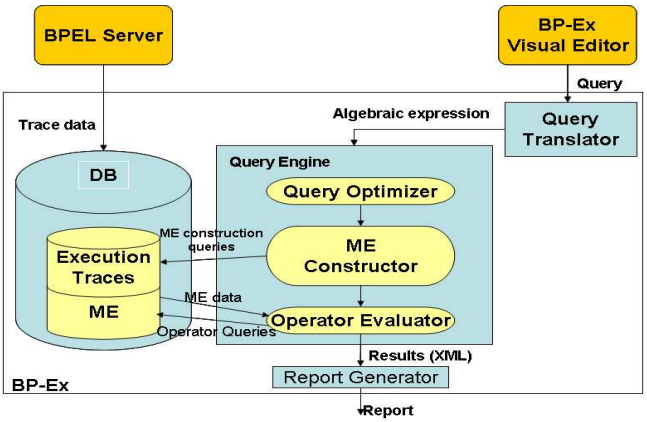

Figure 2: Architecture

Storage and physical operators. The execution traces are stored in a relation database that records the performed activities and their execution order. (Intuitively, the nodes and edges in the execution trace DAG). For each composite activities, the internal execution flow is also recorded (and, recursively, the flow of its composite activities). The matching of the query pattern to the execution traces (and the corresponding ME construction) is performed by a set of SQL queries. To facilitate this SQL-based pattern-matching, each trace activity is assigned a special id, such that given the ids of two activities, one can directly determine whether on precedes the other in the flow and/or belongs to its internal flow. The identification scheme that we use is an adjustment of the XML (tree) node ids of [14] to nested trace DAGs.

Each algebraic operator has two types of possible physical implementation. The first is memory-based and can be used whenever the constructed ME is small enough to fit into main memory. When the ME is large (as often is the case), it is stored in the database, and each operator is then implemented by a set of SQL queries over the stored ME. A dedicated optimization technique that we have developed allows to slice the ME into smaller parts that can fit into main memory, thereby performing a mixed memory/databse-based computation. A cost model, based on statistics gathered from the execution traces repository, is used to choose among the possible physical execution plans.

4. BP-EX SYSTEM ARCHITECTURE

The architecture of BP-Ex is depicted in Figure 2. The demonstration will illustrate each of the components and their interaction. The system runs on Windows XP Professional, JBoss AS 4.0.4. Oracle BPEL Process Manager 10.1.2. with Oracle 10g database. The visual interface is implemented as an Eclipse plug-in, similarly to Oracle BPEL designer.

BPEL Server. To illustrate the operation of the system is we will consider in the demo a set of BPEL business processes used by a consortium offering travel-related services. The travel agency BP runs on a standard (Oracle, in the is case) BPEL server[17]. The server traces the process instances - the activities they perform, messages sent or received by each activity, variable values, etc. This data is logged and stored in a traces repository (implemented using Oracle 10g).

Visual editor. BP-Ex queries are written via the BP-Suite visual editor, in one of two modes: The user can draw the patterns from scratch, using a drag-and-drop items palette. Or, starting from

a specification of a BP P, use a wizard to create queries for P as follows: The user marks the activities of P that she wishes to include in the query. Then by one click a query draft is created, where non selected activities are omitted and the selected ones are connected by edges that reflect their flow and zoom-in relationship in P. The user can then add conditions on the activities, specify the desired type of analytical processing, ask for various reports, make final adjustments, and click a button to save and/or run the query.

Query Translator. The query created by the visual editor is transformed into algebraic representation that is sent to the query engine for evaluation.

Query engine. The query engine employs algebraic rewrite rules to optimize the query. To evaluate the query, an ME (Multi-Embedding) is constructed, summarizing all occurrences of the query pattern in those traces that are of interest to the user. Based on the size of the constructed ME and statistics gathered from the repository, the algebraic operators are mapped to a corresponding physical execution plan. The final step is the generation of the requested reports.

5. DEMONSTRATION SCENARIO

As mentioned above, we will consider in the demo a set of BPEL business processes used by a consortium offering travel-related services. The processes include flight and train reservation, car rental, and credit and accounting services. The Web interface and execution flow of the BP simulates the Yahoo! Travel application, with the business data (prices, schedules, deals, etc.) synthetically generated. To construct a large database of execution traces, we had first asked 20 users to use the system, performing different travel-related tasks, and then used the obtained execution traces as a seed for automatic generation of traces with similar characteristics.

The purpose of our demonstration is two fold. First, looking at the full `BP-Suite` system, we will demonstrate the uniformity and tight coupling of its three sub systems. Second, focusing on the new `BP-Ex` component, we will demonstrate its novel features, from both the UI and the implementation perspectives.

Query formulation. We will first show how the same intuitive graphical query can be used to (1) check that the BP specification obeys certain company policies, (2) perform critical run-time monitoring tasks, and (3) analyze the process execution traces. As a simple example, the query in Figure 1 (ignoring the `OLAP` icon), when interpreted as a query over the specification, checks whether the BP allows reservations to be confirmed without giving first credit card details. The same query when interpreted as a monitoring task (with the `OLAP` icon replaced by an appropriate `Report` icon), can alert at run-time on such a confirmation attempt. Finally, when interpreted as a query over the execution traces, the query identifies all the occurrences of such execution pattern, groups them by user id and sums up the transactions performed by each criminal user. The ease of query formulation will be illustrated in the demo by comparing our graphical query interface to that used by commercial vendors for the *specification* of BPEL business processes (e.g. [17]); there is a tight analogy between how processes are specified and how they are queried.

Together with the audience we will formulate several queries, illustrating the richness and flexibility of the query interface, both for specifying the execution patterns of interest, and for describing the aggregation/reporting to be performed on them

Query execution. We will then follow tightly the execution of the queries. Specifically, we will focus on two queries - a very selective query where the ME is small and fits into main memory, and a less selective one where the ME is very large and has to be stored on disk. We will first show the algebraic representation of the queries before and after optimization, highlighting the relevant rewrite rules. Next we will demonstrate the constructed ME, showing how it factorizes compactly multiple pattern embedding. Finally, we will follow the physical evaluation of the queries. For the two queries, we will examine the possible execution plans and their estimated execution costs, and show the plan selected by `BP-Ex`. We will demonstrate the performance gain of this choice by running each of the possible plans and comparing its running time to that of the selected `BP-Ex` plan. To conclude the demonstration we will explain the cost model employed by `BP-Ex` and how it lead to this optimized choice.

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, and A. Pilberg. Monitoring business processes with queries. In *VLDB*, pages 603–614, 2007.

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

[5] F. Casati, M. Castellanos, U. Dayal, and N. Salazar. A generic solution for warehousing business process data. In *VLDB*, pages 1128–1137, 2007.

[6] U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson. Data integration flows for business intelligence. In *EDBT*, pages 1–11, 2009.

[7] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

[8] J. Eder, G. E. Olivotto, and W. Gruber. A data warehouse for workflow logs. In *Proc. of EDCIS*, pages 1–15, 2002.

[9] X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. of the Int. WWW Conf.*, 2004.

[10] H. Hacigumus. Compliance enforcement for service process flows. In *IEEE SCC*, pages 682–683, 2007.

[11] M. C. Hao, Daniel A. Keim, U. Dayal, and J. Schneidewind. Business process impact visualization and anomaly detection. *Information Visualization*, 5(1):15–27, 2006.

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

[13] ILOG JViews. http://www.ilog.com/products/jviews/.

[14] H. Kaplan, T. Milo, and R. Shabo. Compact labeling scheme for xml ancestor queries. *Theory Comput. Syst.*, 40(1):55–99, 2007.

[15] M. Lam, J., V. B. Livshits, M. Martin, D. Avots, M. Carbin, and C. Unkel. Context-sensitive program analysis as database queries. In *PODS*, pages 1–12, 2005.

[16] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison Wesley, 2002.

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

[18] A. V. Paliwal, N. Adam, and C. Bornhövd. Web service orchestration and verification using msc and cp nets. In *Proc. ACM symp. on Applied computing*, pages 1693–1694, 2007.