Querying Business Processes with BP-QL*

Catriel Beeri
Hebrew University

Anat Eyal
Tel Aviv University

Simon Kamenkovich
Tel Aviv University

Tova Milo
Tel Aviv University

1 Introduction

A business process consists of a group of business activities undertaken by one or more organizations in pursuit of some particular goal. It usually depends upon various business functions for support, e.g. personnel, accounting, inventory, and interacts with other business processes/activities carried by the same or other organizations. Consequently, the software implementing a business processes typically operates in a cross-organization, distributed environment.

Standards facilitate the design, development and deployment of software for such processes. It is common practice to use *XML* for data exchange between business processes, and *Web services* for interaction with remote processes [10]. The recently emerging BPEL standard (Business Process Execution Language [3], also identified as BPELWS or BPEL4WS) takes standardization a big step forward. Developed jointly by BEA Systems, IBM, and Microsoft, BPEL combines and replaces IBM's WebServices Flow Language (WSFL) and Microsoft's XLANG. It provides an XML-based language to describe not only the interface between the participants in a process, but also the *full operational logic* of the process and its *execution flow*. A BPEL specification, once written, can be compiled into executable code that implements the described business process [7].

Declarative BPEL specifications greatly simplify the task of software development for business processes. More interestingly from an information management perspective, they also provide an important new *mine of information*. Consider for instance a user who tries to understand how a particular business, say a travel agency, operates. She may want to find answers to questions like the following:
Can I get a price quote without giving first my credit card details? What should one do to confirm a purchase? What kind of credit services are used by the agency, directly or indirectly, (i.e. by the other processes it interacts with)?

*The research has been supported by the Israel Science Foundation.

**Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005**

Obviously, such queries are of great interest to both individual users and to organizations interested in using or analyzing business processes. Answering them is extremely hard (if not impossible) when the business process logic is coded in a complex program. It is potentially much easier given a *declarative specification* like BPEL. While the BPEL specifications of a travel agency may not be publicly available, those of an organization are available inside the organization. If it cooperates with other organizations, then their relevant BPEL specifications may be available to it as well. Thus, being able to answer such queries, in a possibly distributed environment, is of great practical potential.

To support such queries, one needs a data model, an adequate query language, and an efficient execution engine for it. In this demonstration we present BP-QL, a new query language which allows for a GUI-based intuitive query formulation of business processes specifications, and query execution in a distributed environment.

Demonstration highlights To illustrate the features of BP-QL we will consider a set of business processes used by a consortium offering traveling-related services. The processes include flight and train reservation, car rental, and credit and accounting services. The processes, and their BPEL specifications, reside and operate on three peers (three computers, in our demo). The specifications include the interactions between the various processes.

We will show how the intuitive graphical BP-QL query interface is used to query and analyze processes specification at different levels of granularity: *Fine-grained* queries allow to "zoom in" on all the process components (local as well as remote ones); *coarse-grain* queries consider process components as black boxes and allow for higher level abstraction. 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. [7]); there is a tight analogy between how processes are specified and how they are queried.

The BP-QL prototype system is implemented in Active XML [2] (AXML for short). In AXML documents, some of the XML data is explicit, while other parts are presented intentionally, by means of Web service calls. Such calls are used in our implementation of BP-QL to access process components provided by remote peers. We will show in the demo how BPEL specifications are wrapped and modeled as AXML documents, and how BP-QL queries are com-

piled into XQuery-like queries over such documents. Service call invocations will be tracked and displayed to illustrate query optimization and execution.

The remaining of this paper is structured as follows. Section 2 describes the main principles that guided the design of BP-QL. Section 3 presents the data model and illustrates the BP-QL query language via the demo scenario example. Section 4 considers the system implementation.

2 Design principles

Before presenting BP-QL, let us first highlight two of the main issues that influenced its design.

Querying specifications vs. possible runs. BP-QL users formulate queries about process *specifications* (e.g. "does the specification include these two activities") and not about *possible runs* (e.g. "can there be a run of the system where both activities take place").

This is for two main reasons. First, querying the possible runs of a system is inherently a *verification problem* [4] and is typically of very high complexity (NP-hard even for very simple specifications and undecidable in more general cases [6]). Second, the analysis of runs requires the specification to have a well defined semantics. Unfortunately, BPEL is not based on a formal model [6].

In contrast, as we show below, queries on specifications have a well defined semantics, and can be efficiently performed. Note that querying of specifications in fact "approximates" the querying of runs (e.g. only specifications that contain two given activities may potentially have runs where both occur). Hence, even when runs verification is desired, BP-QL can be used as an efficient means to narrow the search space for the more costly, interpretation dependent verification. It can also be used to select the process parts whose runs should be monitored at execution time [8].

A dedicated language vs. XQuery. A BPEL specification is essentially an XML document. Thus, a natural question is why not simply use XQuery to query it? A key observation is that the BPEL XML format was designed with ease of *automatic code generation* in mind; however, it is extremely inconvenient as far as *querying* is concerned. To express even a very simple inquiry about a process execution flow, one needs to write a fairly complex XQuery query that performs an excessive number of joins. Essentially, ease of querying requires a data model that allows to naturally and easily represent the main features and components of an entity. For business processes, XML cannot serve this role. Furthermore, when querying business processes, users are often interested to retrieve *execution paths* as answers (as for instance in the query "What should I do to confirm my purchase?"). But, XQuery is targeted to return documents *elements*, but not the *paths* leading to them.

In contrast, BP-QL is based on an intuitive model of business processes, an abstraction of the BPEL specification that hides the tedious XML details, along with a graphical user interface that allows for simple formulation of queries over this model. In a sense, it follows the same design principles that guided commercial vendors in the development of graphical editors for business processes specification (e.g. [7]). Execution paths are considered first class objects and can be retrieved, even when involving activities performed on distinct peers.

3 Data model and query language

We informally describe our data model and the BP-QL query language via an example from the demo scenario.

Data Model A process specification includes:

1. the list of activities/services of which the process is composed, including their types and properties,
2. the data used in the process (referred in the sequel as the process *database*), namely the process variables and the input and output parameters for the participating activities/services,
3. a description of the process operational and data flow.

A UDDI [9] style specification is used for (1) and XML for (2). We model the process operational and data flow using a an extended notion of *statechart* [5]. A statechart (see [5]) consists of (possibly nested) states, with transitions between states represented by directed edges labeled by event names. State nesting allows to zoom in on a particular state and describe its internal structure and flow, also as a statechart. Our extensions allow to capture some of the particular aspects of business processes. In particular, to model the operational flow of business processes, we distinguish two particular types of states, which we call *provided operation* and *requested operation*. These describe, resp., the services offered by a process to other processes, and the external services requested by the given process. We also use particular nodes to represent *data elements*, and *data-flow edges* to describe how data are passed around.

Consider for instance the business process depicted in Figure 1. It represents a travel agency. Its two components are the database, and the statechart that specifies its flow. The small circle at the top of the statechart is its entry point. The processes consists of two composite states, namely `planTrip` and `reserveTrip`. Customers of the agency start with the first and can then continue to the second. Inside each composite state, the titles below the icons are the substates names. In `planTrip`, a customer may either search a for rental car or for a flight. The `searchFlights` state is a requested operation, (as indicated by the icon's shape), since to search for flights one needs information about flights from one or more reservation services; these are external to this process. The input to these reservation services, namely the set of airlines that the agency works with, is imported from the `airlines` table in the database when a customer enters the `searchFlights` state. This is indicated by the double arrow (indicating data flow) from the table to the operation. After a customer has selected some flights, these are stored in the database, in `searchResults`. They are thus available for use in `reserveTrip` (this is not detailed in the Figure here). The same holds for selected car rentals (omitted from the Figure). Note that the database

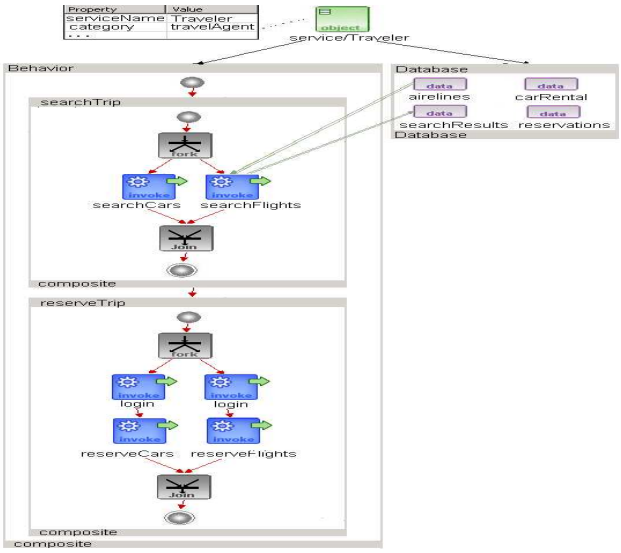

Figure 1: A Travel Agency business process.

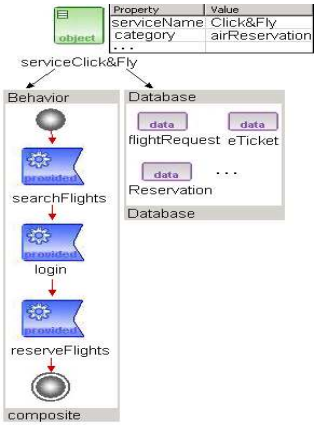

Figure 2: Flights reservation business process.

represents both the agency general data, and a customer's data generated in an execution of the process.

The process in Figure 2 represents a flight reservation service. Here, `searchFlights` is a provided operation (as indicated by the icon's shape), that matches the requested operation of the same name in the agency statechart. A similar relationship holds between the `login` and the `reserveFlights` states, in the two processes.

The BP-QL query language For querying business processes, BP-QL offers *statechart patterns*. Intuitively, these patterns play for statecharts a role analogous to the one that *tree pattern queries* play for XML trees. They describe the pattern of execution/data flow that is of interest to the user.

In addition to standard statechart notation, our patterns use the following special notation: Following the use of "/' and "//" for one and multiple step navigation in XPath and XQuery, we use arrows with *single/double* heads to denote paths in a statechart of length one or more. Similarly, double bounding boxes in a state denote an unbounded "zoom in" into the state's internal specification. *Selection conditions* can be attached to query nodes and edges. Finally, dashed states and edges represent negation.

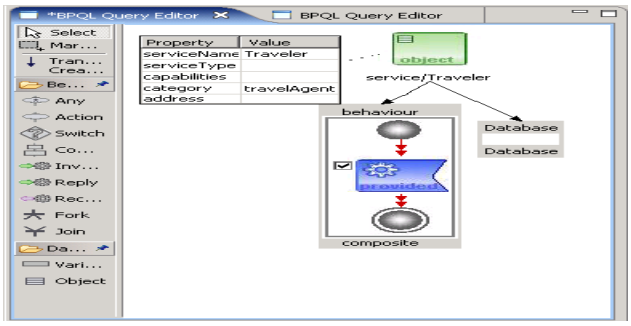

Figure 3: Find provided operations.

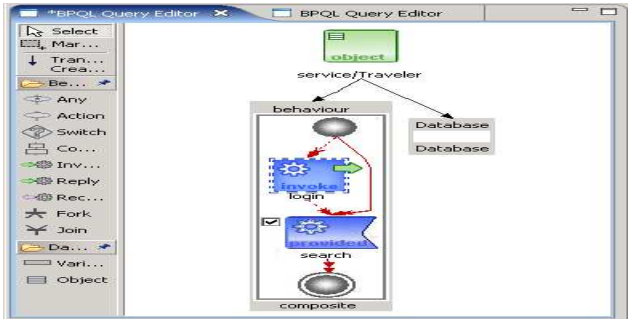

Figure 4: No login.

Four example queries are depicted in Figures 3-6. In each query, the statechart pattern describes the process pattern that a user is looking for. The small check boxes next to the states and edges mark selected states and paths, resp., that the user wants to to retrieve as the query result. The first query, in Figure 3, asks for all the provided operations of the `Travelers` service, at any depth of nesting. The double arrow indicates that operations at any distance from the start state may qualify, the double bounding box denotes unbounded zoom-in, while the shape restricts the query to provided operations. Note that the zoom-in implies that remote operations invoked by this service also qualify. The result (indicated by the small check box) is the set of all qualifying operations. The second query, in Figure 4, asks whether a user of the `Travelers` service

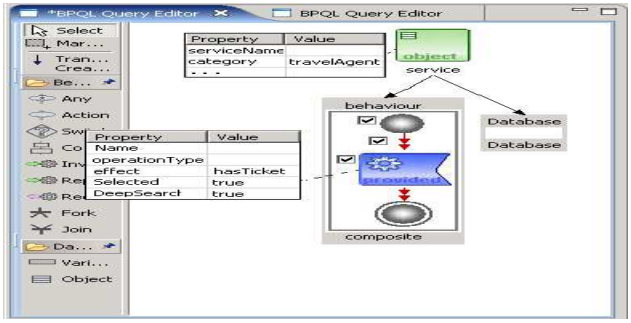

Figure 5: Finding a sequence of operations.

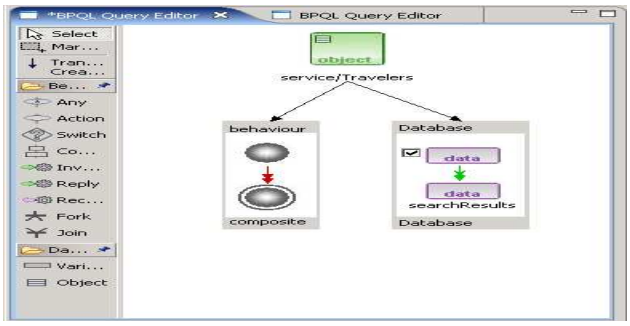

Figure 6: Querying the data flow.

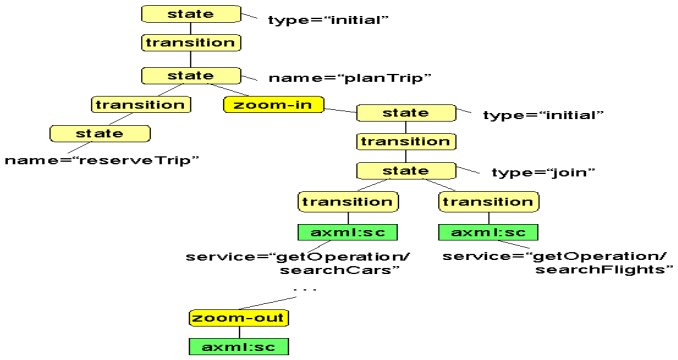

Figure 7: AXML tree.

needs to login for performing a search. Formally, this is expressed by asking: Is there a path of any length that leads to a `search`, and does **not** pass through a `login` request. The *dashed* path here denotes negation. It states that a path to the search activity that passes through a login state, does not exists. The double boundary of the login box indicates that we refer to requests issued by this process or any of the (possibly remote) services it calls, at any dept of nesting. Observe that the same query, without double-boundary boxes, would provide instead a coarse-grained analysis that considers only local requests and views remote services as black boxes. The query in Figure 5 finds the sequences of operations required to buy a flight ticket in processes in the `travelAgent` category. Here, the edge marked by a check-box is bound to any path that leads to a provided operation whose effect includes "hasTickets". The full path, including the start state, the path bound to the edge, and the pointed state are returned as a result. Finally, the query in Figure 6 checks which service data affects the value of the `searchResults` variable.

Query semantics When a query is evaluated, its nodes and edges are viewed as variables and are assigned state (or data) nodes and execution (or data flow) paths, resp., matching the pattern constraints. These are then used to construct the query result. Note that when a process flow contains cycles, the number of paths that may match a given query edge (path variable) may be infinite. BP-QL overcomes this problem by providing a concise finite representation for a possibly infinite set of paths my means of a statechart that may contain cycles.

Partial evaluation In a distributed, cross-organization environment, not all the parties participating in a business process may be willing to expose their BPEL spec. We will show that *partial query evaluation* can still be conducted in such environments, exploiting the available information.

4 Implementation

BP-QL uses the Active XML system [2] as an implementation platform. As mentioned, in AXML documents, some of the XML data is given explicitly, some only intentionally by means of Web service calls. When a query is evaluated over such documents, the calls relevant to the query execution are dynamically invoked, recursive calls are tracked, and only the required data is materialized[1]. In BP-QL we use the AXML services calls to (1) retrieve, when needed, the specifications of remote processes, thus supporting distributed processing, and (2) account for the graph structure of the specification (service calls play here role similar to XML idrefs, with the advantage that they are "traversed" automatically in query evaluation). To see an example consider the (A)XML tree in Figure 7, representing (part of) the BPEL statechart from Figure 1. The *axml:sc* elements represent service calls that retrieve the pointed data.

BP-QL queries are compiled into XQuery queries over these documents. While one could believe that a translation to XQuery should be relatively easy, this turns out not to be the case. First, XQuery does not support path variables. Additionally, it seems to lack features (such as *regular path expressions*) that are necessary for expressing $(/zoom-in)^*$, and $(/zoom-out)^*$. Our strategy (details omitted) uses certain structural ids in the AXML representation, and uses XQuery for initial processing, followed by post-processing for obtaining the desired results.

Each peer has a graphical interface that allows to specify business processes and query them. We will demonstrate how BPEL specs are wrapped and represented by AXML documents, how they are represented with our data model, and how BP-QL queries are compiled and evaluated.

References

[1] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy Evaluation of Active XML Queries. In *Proc. of ACM SIGMOD*, 2004.

[2] Active XML. http://activexml.net/.

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

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

[5] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[6] S. Narayanan and S. McIlraith. Analysis and simulation of web services. *Compute Networks*, 42:675–693, 2003.

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

[8] D. M. Sayal, F. Casati, U. Dayal, and M. Shan. Business Process Cockpit. In *Proc. of VLDB*, 2002.

[9] Universal Description, Discovery and Integration (UDDI). http://www.uddi.ord.

[10] The World Wide Web Consortium. `http://www.w3.org/`.