Querying and Monitoring Distributed Business Processes *

Tutorial

Tova Milo Daniel Deutch
Tel Aviv University
{milo,danielde}@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 usually operates in a cross-organization, distributed environment 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), 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. Processes execution is traced, and their run-time behavior can be recorded in standard XML formats.

These new standards not only simplify software development, but, more interestingly from an information management perspective, they also provide an important new *mine of information*. Queries about the BPs, that were extremely hard (if not impossible) to evaluate when the BP logic was coded in a complex program are now potentially much easier given a declarative specification of the BP. Furthermore, sophisticated querying, that interleaves static analysis of the BP specification with run-time process monitoring, can now be used for a variety of critical tasks such as fraud detection, SLA (service level agreement) maintenance, and general business management. This provides an essential infrastructure for companies to optimize their business processes, reduce operational costs, and ultimately increase competitiveness.

Our goal here to study the new possibilities that this new generation of BPs brings to process querying and monitoring. We first give, in section 2, a brief survey of the new generation BP technology and the challenges it raises. Then in the next two sections we highlight the main properties of existing approaches for the querying (Section 3) and monitoring (Section 4) of BPs, and the consequent challenges.

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

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-306-8/08/08

2. BACKGROUND AND MOTIVATION

A BP usually depends upon various business functions for support, e.g. personnel, accounting, inventory, and interacts with other BPs/activities carried by the same or other organizations. Consequently, the software implementing such BPs typically operates in a cross-organization, distributed environment. It is a common practice to use XML for data exchange between BPs, and *Web services* for interaction with remote processes [37]. The recent BPEL standard (Business Process Execution Language [8], also identified as BPELWS or BPEL4WS), developed jointly by BEA Systems, IBM, and Microsoft, combines and replaces IBM's WebServices Flow Language (WSFL) [24] and Microsoft's XLANG [39]. 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*. Because of the complexity of the BPEL syntax, commercial vendors offer systems that allow to design BPEL specification via a visual interface, using a conceptual, intuitive view of the process, as a graph of data and activity nodes, connected by control flow edges. Designs are automatically converted to BPEL specifications, which in turn can be automatically compiled into executable code that implements the described BP [29].

As mentioned above, declarative BPEL specifications greatly simplify the task of software development for BPs. More interestingly from an information management perspective, they also provide an important new mine of information [36, 31]. Consider for instance a user who tries to understand how a particular business, say a Web auctioning system, operates. She may want to find answers to questions such as: Can I place a bid without giving first my credit card details? What should one do to confirm a purchase? What kind of credit services are used by the auctioning system, directly or indirectly, (i.e. by the other processes it interacts with)? Answering such queries becomes feasible given a declarative BP specification, like BPEL. For an organization that has access to its own BPEL specifications, as well to those of cooperating organizations, the ability to answer such queries, in a possibly distributed environment, is of great practical potential.

To support such queries, one needs an adequate query language, and an efficient execution engine for it. *We argue that it is essential to develop query languages that allow for an intuitive formulation of queries on BP specifications, and*

efficient query evaluation, in a distributed cross-organization environment [5, 14].

A BP (BPEL) specification describes the *potential behavior* of the BP. An *instance* of the BP 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 that they perform, the messages sent or received by each activity, the values of variables, performance metrics - and send this information as events, in standard XML format, to log flies or runtime monitoring systems. These are often called in the literature BAM (Business Activity Monitoring) systems.

To get some intuition about the type of monitoring that a given BP may require, consider again the Web auctioning system mentioned above. The system administrator would like, among others, to guarantee fair play, detect frauds, and track services usage and performance. This may be achieved by monitoring the running processes: she can ask, for instance, to be notified whenever auctioneers cancel bids too often, or when buyers attempt to confirm bids without giving first their credit details. She may also want to be informed whenever the average response time of the database, in a given service, passes a certain threshold, so that she can fix the problem or switch to a backup database. In general, monitoring encompasses the tracking, at run time, of particular patterns in the execution path of individual processes or in the interaction between different processes, as well as the provision of statistics on the performance of one or more processes. One can also query, posteriori, the BP logs to mine such activity patterns.

Observe that the querying of the *potential behavior* of BPs and the monitoring of the *actual run* of process instances are complementary. Queries on the specification can be used to focus on (the parts of) the BPs that require monitoring/logging. Conversely, run-time monitoring/logging can be used to complement the analysis of process properties that cannot be statically determined by querying the specification.

We argue that is important to develop frameworks that gracefully combine the querying of BP specifications, run time behavior, and logs, for a comprehensive BP analysis [6].

Since BPs in general, and BPEL ones in particular, are promised such a brilliant future, we believe it is important to develop a solid foundation for querying and monitoring such processes, thereby providing the essential infrastructure for companies to optimize business processes, reduce operational costs, gain real-time visibility into key performance indicators, and ultimately increase competitiveness.

3. QUERYING BP SPECIFICATIONS

Depending on the context, a user may be interested to query (1) the structure of the specification or (2) the behavior of the process defined by it. We refer to such queries as *structural* and *behavioral* queries, resp. Consider for example a BPEL specification describing the auctioning system in our running example. A software engineer may wish to query the specification to find process components that follow a certain code pattern, say a loop that contains an *IncreaseBid* operation; this is an example of a structural query. A system analyst may, on the other hand, wish to know whether the system allows users to perform an unbounded sequence of bid increments, regardless of how such a sequence is implemented (e.g. with a loop, recursion, or else); this is an example for a behavioral query. These two classes of queries are typically categorized in the literature by their invariance to bisimulation. Intuitively, two specifications are bisimilar if each one's execution can be considered as a simulation of an execution of the other. A query language (or logic) L is invariant to bisimulation if an evaluation of any formula f of L on a system R is equivalent to the evaluation of f on any R' that is bisimilar to R.

Previous research had addressed each of these classes of queries separately, as follows.

Program verification. Works in the area of program verification focus on *behavioral queries*. There has been a vast amount of work in the general area of program analysis and verification (see e.g. [11, 23] for a sample), and more specifically in the analysis of interactions of composite web services and BPEL processes [16, 11, 15, 7, 1]. These works are typically based on modal logics [26] such as LTL, CTL(*) and μ-calculus, which are all bisimulation invariant. Queries, formulated as logic formulas, test if the runs of the program satisfy a behavioral property. The verification of the behavioral properties is typically of very high complexity (from NP-hard for very simple specifications to undecidable in the general case [28]). Dedicated optimization techniques and data structures have been developed to accelerate the process (e.g. [32, 23]).

Database query languages. BP specifications may be abstractly viewed as a set of *nested* graphs, possibly with recursion: the graphs structure captures the execution flow of the process components; the nesting comes from the fact that the operations/services used in a process are not necessarily atomic and may have a complex internal structure (which may itself be represented by a graph); recursion may exist due to mutual calls. Database research offers query languages for semi-structured data in general, and for tree- and graph-shaped data in particular. But typical database query languages are sensitive to the exact graph structure and are not bisimulation invariant [12, 30], hence express only *structural queries*. They also typically consider only flat graphs. This line of research on querying XML and semi-structured data led to the development of standard query languages, an array of query optimization techniques for query evaluation in both centralized and distributed environments [27, 19, 38, 34, 10, 5, 2], and the identification of language fragments with good balance between expressibility and low complexity [17].

Overall, each of these lines of works have some very important merits and some limitations. We believe that an effective solution can be achieved by combining the best features of the two paradigms: structural querying and efficiency of query processing and distributed data management of database systems, with behavioral analysis and dedicated data structures from program verification [13]. We envision a system where users can issue declarative queries on both the structure of BP specifications and the processes expected behavior, with efficient query processing and transparent management of distributed data.

4. MONITORING BPS

BP Management systems allow to trace process instances - the activities that they perform, the messages sent or received by each activity, the values of variables, performance metrics - and to send this information as events, at runtime, to *monitoring* tools (often called in the literature BAM – Business Activity Monitoring) [33, 9, 35, 4, 20, 21]. Events are reported in standard XML formats to enable interoperability. Typical monitoring systems are composed of three layers: a layer that absorbs the stream of events coming from the BP execution engine, a processing and filtering layer that selects relevant events/data and automatically triggers actions, and a dashboard that allows users to follow the process progress, view custom reports and statistics on the processes and send alerts. The events can also be logged and be available to be queried/mined posteriori.

While rather powerful, existing tools are dedicated to the monitoring and mining of run-time events. A process analysis, that interleaves static analysis of the BP specification with run-time analysis of the events stream generated by the process instance, is not possible, using existing tools.

In contrast, the use of uniform query languages for querying both static and streamed data is common in database system [22, 3]. We have mentioned above that the events sent to BP monitoring systems have standard XML format and that BP (BPEL) specifications are also written in XML. A natural question is why not use XQuery, coupled with some existing XML stream-processing engine (e.g. [18, 25]), for the task? We have already mentioned above the limitation of database query languages in general, and XQuery in particular, for the querying of BP specifications. XML-stream processing engines are also not the best fit here for monitoring the events trace. A key observation is that the XML elements in this stream each describe an individual event. 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 of such elements and is difficult (if not impossible) to handle by existing streaming engines. Furthermore, even if a more query-friendly nested XML representation, that reflects the flow, had been chosen for the data, standard XML stream processing would still not be adequate for the task: XML stream engines manage tree-shaped data and not DAGs (directed acyclic graphs), which is the typical structure of processes trace. More importantly, they expect to receive the tree elements in document order (i.e. from left to right) and process sibling branches sequentially. But the events flow in BPs does not necessarily follow this order: events of parallel activities may interleave. Deferring the processing of incoming events of a given process branch until all the events of its sibling have been processed may cause an unnecessary and significant delay. A dedicated parallel processing is required here.

Here again, we believe that an adequate solution may be achieved by combining the best features of the existing technologies, enriching them where needed, to obtain a uniform framework for querying BP specs, run time behavior, and logs. This will allow to use specifications analysis to identify process parts that require monitoring/logging and, conversely, exploit monitoring/logging to clarify issues that could not be determined statically.

5. REFERENCES

[1] S. Abiteboul, Zoe Abrams, Stefan Haar, and T. Milo. Diagnosis of asynchronous discrete event systems: datalog to the rescue! In *Proc. of ACM PODS*, 2005.

[2] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic xml documents with distribution and replication. In *Proc. of ACM SIGMOD*, 2003.

[3] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[4] BEA. Weblogic application server. http://www.bea.com.

[5] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In *Proc. of VLDB*, 2006.

[6] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *Proc. of VLDB*, 2007.

[7] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *Proc. of VLDB*, 2005.

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

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

[10] L. Chen, A. Gupta, and E. Kurul. Stack-based Algorithms for Pattern Matching on DAGs. In *Proc. of VLDB*, 2005.

[11] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[12] M. Consens and A. Mendelzon. The g+/graphlog visual query system. In *Proc. of ACM SIGMOD*, page 388, 1990.

[13] D. Deutch and T. Milo. Querying structural and behavioral properties of business processes. In *Proc. of DBPL*, 2007.

[14] D. Deutch and T. Milo. Type inference and type checking for queries on execution traces. In *Proc. of VLDB*, 2008.

[15] A. Deutsch, M. Marcus, L. Sui, V. Vianu, and D. Zhou. A verifier for interactive, data-driven web applications. In *Proc. of ACM SIGMOD*, 2005.

[16] E.Clarke, O. Grumberg, and D. Long. Verification Tools for Finite State Concurrent Systems. In *A Decade of Concurrency-Reflections and Perspectives*, volume 803, pages 124–175. Springer-Verlag, 1993.

[17] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[18] A. K. Gupta and D. Suciu. Stream Processing of XPath Queries with Predicates. In *Proc. of ACM SIGMOD*, 2003.

[19] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer-data management system. *Trans. on Knowledge and Data Engineering*, 16(7):787–798, 2004.

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

[21] Ilog jviews. http://www.ilog.com/products/jviews/.
[22] N. Koudas and D. Srivastava. Data Stream Query Processing. In *Proc. of VLDB*, 2003.
[23] M. Lam, J., V. B. Livshits, M. Martin, D. Avots, M. Carbin, and C. Unkel. Context-sensitive program analysis as database queries. In *PODS*, 2005.
[24] F. Leymann. Web Services Flow Language (WSFL) 1.1, May 2001. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
[25] X. Li and G. Agrawal. Efficient Evaluation of XQuery over Streaming Data. In *Proc. of VLDB*, 2005.
[26] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.
[27] J. McHugh and J. Widom. Query optimization for XML. In *The VLDB Journal*, pages 315–326, 1999.
[28] S. Narayanan and S. McIlraith. Analysis and simulation of web services. *Compute Networks*, 42:675–693, 2003.
[29] Oracle BPEL Process Manager 2.0 Quick Start Tutorial. http://www.oracle.com/technology/products/ias/bpel/index.html.
[30] J. Paredaens, P. Peelman, and L. Tanca. G-log: A graph-based query language. *IEEE Trans. Knowl. Data Eng.*, 7(3):436–453, 1995.
[31] D. Roman and M. Kifer. Reasoning about the behavior of semantic web services with concurrent transaction logic. In *Proc. of VLDB*, 2007.
[32] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *POPL*, 1999.
[33] D. M. Sayal, F. Casati, U. Dayal, and M. Shan. Business Process Cockpit. In *Proc. of VLDB*, 2002.
[34] D. Suciu. Distributed query evaluation on semistructured data. *Database Systems*, 27(1):1–62, 2002.
[35] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst. The prom framework: A new era in process mining tool support. In *ICATPN*, pages 444–454, 2005.
[36] M. Vrhovnik, H. Schwarz, O. Suhre, B. Mitschang, V. Markl, A. Maier, and T. Kraft. An approach to optimize data processing in business processes. In *Proc. of VLDB*, 2007.
[37] The World Wide Web Consortium. `http://www.w3.org/`.
[38] Y. Wu and H. Jagadish. Structural join order selection for xml query optimization, 2003.
[39] XLANG: Web Services for Business Process Design. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.