Querying Future and Past in Business Processes *

Daniel Deutch Tova Milo

Tel Aviv University
{danielde,milo}@post.tau.ac.il

Abstract

A business process (BP for short) consists of a group of business activities undertaken in pursuit of some particular goal. Analysis of BPs bears two main flavors, namely analysis of future *and* past *executions. We intuitively explain these analysis goals and the models and algorithms employed to achieve them.*

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 BPEL [5] 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 (logged), and their run-time behavior can be recorded in standard XML formats.

These 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 business rules were coded in a complex program, are now potentially much easier, for a declarative specification of the BP. Furthermore, sophisticated querying, that interleaves static analysis of the BP specification with queries over execution traces, 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 to both companies and customers: the former may optimize their business processes, reduce operational costs, and ultimately increase competitiveness. The latter may be presented with personalized analysis of the process, allowing them to make an optimal use of it.

For instance, consider a Business Process of an on-line store, that suggests electrical products of various kinds, brands and vendors. The web-site owner, on one hand, may wish to make sure that some business logic is kept, e.g. that no customers can make a product reservation without logging-in with their credit card number first; or in identifying the DVD brand that is most popular among customers buying a certain TV brand. The users, on the other hand, may wish to analyze the BP for identifying compatible TV and DVD of the lowest total price, or the most common choice of combined products.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

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

In general, analysis of such BPs bears two main flavors. First, analysts are interested in analyzing executions that occurred in the *past*, for instance to identify trends, to make sure that business logic is maintained, etc. We note that executions are usually logged, forming *execution traces* that are kept in a repository. Thus, analysis over past executions is translated into queries over traces. There are two cruxes here: first, the size of a typical execution traces repository is extensively large, calling for query optimization techniques. Second, the traces often contain only partial information on the activities that were performed at run time, due to confidentiality, lack of storage space, etc. Thus query evaluation must be performed under terms of uncertainty.

The second flavor of analysis considers *future* executions. Namely, given the static BP specification, this kind of analysis, again operating under terms of uncertainty, aims at predicting the behavior of future users, e.g. to characterize *common* behavior of users, or to identify executions where the total induced cost to the customer is the cheapest, etc. This latter kind of analysis is in fact a *top-k* analysis, as it aims to find the k "best" execution flows, under some weighting function, and the main difficulties here stem from (1) the fact that the number of possible execution flows is very large, or even infinite in presence of recursion and (2) that the weight (e.g. likelihood, monetary cost, etc.) induced by actions made during the flow (e.g. product purchase), may be inter-dependent (due to probabilistic dependency, combined deals etc.).

To enable such reasoning, we first define models for capturing Business Process specifications, their execution flows and traces. These models should account for partial information and uncertainty of various flavors. Second, we define an intuitive query language that allows to rapidly form queries of interest over BP execution flows and traces. Third, we provide algorithms that allow for efficient query evaluation over BPs/execution traces under these models of uncertainty, and forth, we develop implementations that exploit these sound theoretical foundations for practical needs. We Intuitively describe here some of the main models and results.

2 Models

We give next a brief intuitive review of the main models standing at the center of our research on Business Processes analysis, and refer the reader to [12, 13] for precise definitions.

BP specifications and their executions. A BP specification is modeled as a set of node-labeled DAGs. Each DAG, intuitively representing a function, has a unique *start* (*end*) node with no incoming (outgoing) edges. Nodes are labeled by activity names and directed edges impose ordering constraints on activities. Activities that are not linked via a directed path are assumed to occur in parallel. The DAGs are linked through *implementation relationships*: an activity a in one DAG is realized via another DAG. We call such an activity *compound* to differentiate it from *atomic* activities having no implementations. Compound activities may have multiple possible implementations; the choice of implementation is controlled by a condition over user choices, variable values, etc., referred to as a *guarding formula*. A distinguished DAG, containing a single activity, stands as the BP root.

Figure 1(a) shows an example BP specification. The root S_0 has precisely one activity named `ShoppingMall`. The latter has as its implementation the DAG S_1, which describes a group of activities comprising user login, the injection of an advertisement, the choice of a particular store, and the user exit (possibly by paying). Within S_1, `Login` and `chooseStore` are *compound* activities; the `Login` activity has two possible implementations S_2 and S_3; the idea is that exactly one formula is satisfied at run-time, e.g., the user either logins as a regular or premium user and thus `Login` is implemented either by S_2 or S_3 respectively. Then the user chooses to pay with Mastercard or Visa, and authentications checks are made according to her identity (regular or premium), etc. Note that the specification is recursive as e.g. S_9 may call S_1.

An *execution flow* (abbr. EX-flow) of such BP is modeled as a nested DAG reflecting the execution order and implementation relation. Exactly a single implementation is chosen for each compound activity node. We model each activity occurrence by two nodes, the first (second) standing for its *activation* (*completion*) point. The chosen implementation appears in-between these two nodes (connected by specially marked *zoom-in edges*).

An example EX-flow is given in Figure 1(b). Regular (dashed) arrows stand for flow (zoom-in) edges. The user here logins as a regular customer and pays with a *Visa* Credit Card, then shops at the *BestBuy* store. There,

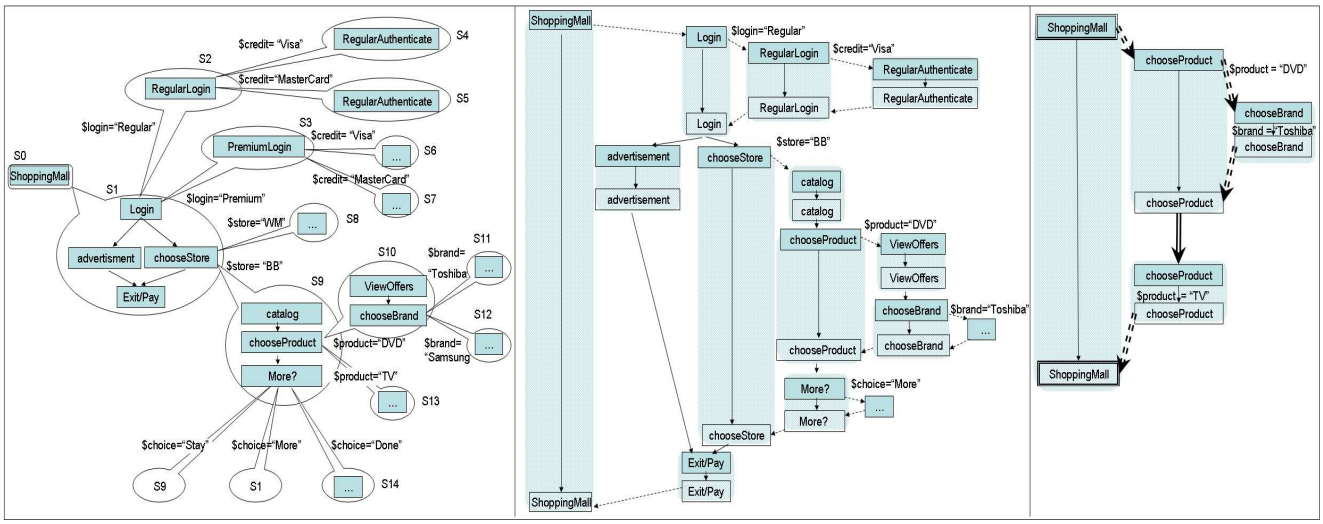

Figure 1: (a) Business Process (b) Execution Flow (c) Execution Pattern

she chooses to look for a *DVD player* and selects one by *Toshiba*, and continues shopping at the same store (we omitted the further purchases from the figure). Finally she exits and pays.

Tracing Systems. Tracing systems are employed for recording execution flows of Business Processes, and may vary in the amount of information that they record on the flow. In general, one can distinguish three families of tracing systems with decreasing amount of information: (i) *naive* tracing provides a complete record of the activation/completion events of all activities that had occurred during the EX-flow, (ii) *semi-naive* tracing where the activation/completion events are all recorded, but possibly with only partial information about their origin activity. For instance, such tracing system may record all login activities (Premium and Regular) as a generic `Login` name (and similarly for `Authenticate`), thus removing track of the different treatment of Premium and Regular clients, and (iii) *selective* tracing where, additionally, events for some selected subset of activities are not recorded at all. Such system may, for instance, completely omit the occurrences of login-related activities, thus removing all record of the fact that there are two types of users in this system.

Queries. Queries select EX-flows of interest using *execution patterns*, an adaptation of the tree/graph patterns offered by existing query languages for XML/graph-shaped data [9], to BP nested DAGs. Execution patterns may be uniformly interpreted as queries over past or future executions. An execution pattern is a nested DAG of shape similar to that of an EX-flow, but its edges may be either regular, i.e. match a single edge in the EX-flow, or transitive, i.e. match a path. Similarly, activity pairs may be regular or transitive for searching only in their direct implementation or zooming-in inside it, resp. Activity nodes may be marked by a special "Any" symbol, and then may be matched to BP nodes with any label, and finally some query part may be marked as output and is projected out (the query language may be enhanced by joins, negation etc. [12]). An example query is given in Fig. 1(c). The double-lined edges (double-boxed nodes) are *transitive* edges (activities). The query looks for EX-flows where the user chooses a DVD of brand Toshiba (possibly after performing some other activities, corresponding to the transitive edges), then chooses also a TV (of any brand). The `ShoppingMall` activity is transitive indicating that its implementation may appear in any nesting depth; `chooseProduct` is not transitive, requiring the brand choice to appear in its direct implementation.

3 Querying Future and Past Executions

We have reviewed the models standing at the center of our research on analyzing Business Processes, and we next (informally) define several main research problems in this context and give intuition for their solutions.

Querying Future Executions Recall that a BP specification defines a set of possible executions. We studied in [3] query evaluation over BP specifications, selecting execution flows of interest. The set of query results may be infinite, due to recursion. Continuing with our running example, there are infinite number of flows ending with the choice of Toshiba DVD, as the user may first make any sequence of choices and selections.

Thus the evaluation algorithm obtains a compact representation of all results, describing this recursive nature of qualifying flows. We note, however, that the query results are not equally interesting, and users are thus interested only in some subset of these, namely the *top-k weighted flows*, according to some weight function that fits the user interests. We thus introduce in [13] a refined, weighted model for execution flows of Business Processes, and compute the *top-k* execution flows of the given process, out of these conforming to the user query. The weighted model bears the following ingredients: first, we define a weight function $cWeight$ (corresponding to e.g. products prices, popularity of a link, etc.) over all possible implementation choices of compound activities; observe that the $cWeight$ of a given choice may vary at different points of the EX-flow and may depend on the course of the flow so far and on previous choices, e.g. the likelihood (price) of choosing a product may depend on previously purchased products, registration to membership club etc. Thus $cWeight$ accounts not only for the choice itself but also information about the history of the EX-flow thus far. Then, we aggregate $cWeight$ values of choices throughout a flow to obtain the weight of the entire flow (denoted $fWeight$, for flow weight). Following common practice [20], we require the aggregation to be monotonic w.r.t. the progress of the flow.

Results. We have shown that the extent to which the $cWeight$ of a given choice is dependent on the preceding choices affects the complexity of our problem. We use the term "history size" to measure this extent. In the general case where the history size is unbounded, top-k query evaluation is undecidable. Fortunately, such case is very rare, and moreover studies on the behavior of typical Web applications indicate this size to be relatively small (approximately 4) [28]. The complexity of our algorithm is then exponential in this (small) size (but we show this is unavoidable, unless P=NP), polynomial in the BP size, and linear in the output size.

We have also examined *optimality* properties of top-k algorithms for BP flows, and found that with plausible assumptions over $fWeight$, intuitively corresponding to "how strongly monotone" it is, one may provide (instance) optimal [20] algorithms for top-k query evaluation. We refer the reader to [14] for details.

Practical Applications. We have exemplified some of the practical applications of possible future flow analysis in [14], where the theoretical background explained above was exploited to design `ShopIT` (ShoppIng assitanT), a system that assists on-line shoppers by suggesting the most effective navigation paths for their specified criteria and preference. When the user starts her navigation in the site, she specifies her constraints and her weighting function of interest, and have the system compute and propose (an initial set of) top-k ranked navigation flows, out of these conforming to the constraints. The user then continues her navigation taking into account the presented recommendations, but may also make choices different than those proposed by the system, in the latter case `ShopIT` adapts its recommendations to the actual choices made by the user.

Querying Past Executions So far we have considered analysis of possible executions that have not happened yet. Naturally, much information can be also obtained from executions that had occurred in the past. Analysis of such information may be either done at run-time, *monitoring* [4] the execution, or over repositories of execution traces [12]. The latter kind of analysis is often done in two steps: the repository is first queried to select portions of the traces that are of particular interest. Then, these serve as input for a finer analysis that further queries and mines the sub-traces to derive critical business information [29]. Not surprisingly, *type information*, i.e., knowledge about the possible structure of the queried (sub-)traces, is valuable for query optimization [4]. Its role is analogous to that of XML schema for XML query optimization: it allows to eliminate redundant computations and simplify query evaluation. Such type information is readily available, as the BP specification, for the original traces, but not for the intermediary traces selected by queries. This calls for *Type Inference*. When the analysis tool expects particular data type, we would also like to verify that the sub-traces selected by queries conform to the required type. Such *Type Checking* is thus a second challenge.

An additional kind of queries over past executions considers *recovery* of the flow that is most likely to actually happened at run-time, given a partial trace. There are two practical cases to consider here: first, the simpler case where the tracing system itself, e.g. which activities are omitted or renamed are known. Note that there may still be (infinitely) many origins to a given log. Second, the tracing system itself may be unknown, in which case there may also be exponentially many tracing systems to consider.

Results. We showed in [12] that the less detailed (and thus less restrictive) the execution traces are, the more efficient type inference can be: it can be done in time polynomial in the size of the input type (with the exponent determined by the size of the query) for selective trace types, but may require time exponential in the size of input type (even for small queries) with semi-naive trace types, and may not be possible at all if all trace types are naive. This signals *selective trace types* as an "ideal" type system for BP traces, allowing both flexible description of the BP traces as well as efficient type inference. Type checking, on the other hand, incurs exponential data complexity for (semi-)naive trace types , and is undecidable for selective trace types. This indicates that static type checking is probably infeasible, and calls for run-time analysis [4].

Retrieval of execution flows given their trace was studied in [13], where we show that our query evaluation algorithms can be adapted to retrieve the most likely flow given a trace. We then consider the case of an unknown tracing system, and avoid enumeration of the exponentially many tracing systems by proving a *small world* theorem, showing that only a polynomial number of representative options need to be tested.

Practical Applications. We have demonstrated in [4] a query language and system for monitoring business processes, that allows users to visually define monitoring tasks, using a simple intuitive interface similar to those used for designing BPEL processes. The monitoring tasks are translated to very efficient BPEL processes that run on the same execution engine as the monitored processes.

4 Related Work

First, let us consider our choice of data model and query language. These are argued [12] to be more intuitive for BP developers than e.g. temporal logics and process algebras, as they are based on the same graph-based view used by commercial vendors for the specification of BPs.

A variety of formalisms for (probabilistic) process specifications exist in the literature, with applications in Verification [19], Natural Language Processing, Bioinformatics, etc. Among those, we mention Hidden Markov Models (HMMs) [18], (Probabilistic) Recursive State Machines (PRSMs) [2], and (Stochastic) Context Free (Graph) Grammars (CFG, CFGG) [10]. While HMM extends Finite State Machines, PRSMs and SCFGG describe nested structures similar to that of BPs. There are several differences between works on these models and analysis of BPs. First, in terms of expressive power, probabilistic variants of these models typically assume independencies (markov property, context freeness) between probabilistic events. Second, most of the analysis works over such processes (e.g. [15, 16, 6]) use temporal logic, which may not capture our query language. Intuitively, this is because our query language bears *structural features* (allowing e.g. to capture graph homomorphism). In contrast, work on querying CFGGs [10] generally uses strongly expressive (structural) logics such as MSO (Monadic Second Order Logic), incurring high evaluation complexity. Also note the analogy between Naive (Semi-naive, Selective) trace types and *bracketed* (*parenthesis* [25], *context free* [31]) string languages.

Type Checking and Type Inference, discussed here for BP execution traces, are well studied problems in functional programming languages for database queries [27, 7], and for XML [26]. The unique structure of BP traces incur additional difficulties (e.g. in contrast to XML, here type checking is harder than type inference).

We have also discussed above top-k queries for execution flows of BPs. Top-k queries were studied extensively in the context of relational and XML data [22]. Notably, [20] presented an instance-optimal algorithm for top-k queries that aggregate individual scores given to joining tuples. Difficulties specific to the BP settings are that (1) the size of a given flow, thus the number of aggregated scores, is unbounded (2) the particular properties of the weight functions are unique to EX-flows and (3) the number of items (EX-flows) that are ranked is infinite. Note that while an infinite setting also appears in top-k queries over *streamed* data [24], works in this context aggregate over a *bounded size sliding window*, whereas we consider aggregation over flows of unbounded size.

Ranking by likelihood was also studied in several other settings, e.g. *Probabilistic Databases* (PDBs) [11, 30] and *Probabilistic XML* [1, 23]. For example, [30] and [23] study the problem of retrieving the top-k query results for queries over PDBs and Probabilistic XML, resp. Note that in contrast to relational data and XML, our model for BP flows allows representation of an *infinite* number of items, out of which the top-k are retrieved.

Last, we briefly mention a complementary line of tools whose input is a set of run-time generated *traces*

(logs), and may generate a probability distribution over the events affecting the flow; such distribution may then serve as input for our analysis. The common OLAP (online analytical processing)-style analysis [17] offers users various multi-dimensional views of data, and correlations in-between. The Business Process Intelligence (BPI) [21] project is another branch of the work on analyzing execution flows, inferring causality relationships between execution attributes using data mining techniques such as classification and association rule mining. Such retrieved relationships may be used as input to our analysis.

5 Conclusion

We have depicted here models and algorithms for capturing and analyzing Business Processes and their past and future executions, and demonstrated that declarative languages for specifying and querying such processes allow for important analysis and optimization tasks, that were not possible in the absence of such languages. Further challenges include, among others, extensions of the query language to include additional useful features such as negation, joins, etc.; considering other settings for top-k analysis; and designing further practical applications.

References

[1] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in xml. In *Proc. of EDBT*, 2006.

[2] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4), 2005.

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

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

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

[6] T. Bultan, J. Su, and X. Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1), 2006.

[7] J. Bussche, D. Gucht, and S. Vansummeren. A crash course on database queries. In *Proc. of PODS*, 2007.

[8] F. Casati, M. Castellanos, N. Salazar, and U. Dayal. Abstract process data warehousing. In *Proc. of ICDE*, 2007.

[9] D. Chamberlin. Xquery: a query language for xml. In *Proc. of SIGMOD*, 2003.

[10] B. Courcelle. The monadic second-order logic of graphs. *Inf. Comput.*, 85(1), 1990.

[11] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. of VLDB*, 2004.

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

[13] D. Deutch and T. Milo. Top-k projection queries for probabilistic business processes. In *Proc. of ICDT*, 2009.

[14] D. Deutch, T. Milo, and T.Yam. Goal-oriented web-site navigation for on-line shoppers. In *Proc. of VLDB*, 2009.

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

[16] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proc. of PODS*, 2006.

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

[18] Y. Ephraim and N. Merhav. Hidden markov processes. *IEEE Trans. Inf. Theory*, 48(6), 2002.

[19] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *Proc. of TACAS*, 2005.

[20] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.

[21] D. Grigori, F. Casati, M. Castellanos, M.Sayal U .Dayal, and M. Shan. Business process intelligence. *Comp. in Industry*, 53, 2004.

[22] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[23] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic xml. In *Proc. of VLDB*, 2007.

[24] N. Koudas and D. Srivastava. Data stream query processing: A tutorial. In *Proc. of VLDB*, 2003.

[25] R. McNaughton. Parenthesis grammars. *J. ACM*, 14(3), 1967.

[26] T. Milo and D. Suciu. Type inference for queries on semistructured data. In *Proc. of PODS*, 1999.

[27] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[28] P. L. T. Pirolli and J. E. Pitkow. Distributions of surfers' paths through the world wide web: Empirical characterizations. *World Wide Web*, 2(1-2), 1999.

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

[30] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proc. of ICDE*, 2007.

[31] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.