Journal of Computer and System Sciences 78 (2012) 583–609

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A structural/temporal query language for Business Processes

Daniel Deutch [a,*], Tova Milo [b]

[a] Ben Gurion University, Israel
[b] Tel Aviv University, Israel

ARTICLE INFO

Article history:
Received 29 August 2010
Received in revised form 30 August 2011
Accepted 12 September 2011
Available online 14 September 2011

Keywords:
Business Processes
Query languages

ABSTRACT

A Business Process consists of multiple business activities, which, when combined in a flow, achieve some particular goal. These processes usually operate in a distributed environment and the software implementing them is fairly complex. Thus, effective tools for analysis of the possible executions of such processes are extremely important for companies (Beeri et al., 2006, 2007 [4,5]); (Deutch and Milo, 2008 [13]); these tools can allow to debug and optimize the processes, and to make an optimal use of them. The goal of the present paper is to consider a formal model underlying Business Processes and study query languages over such processes. We study in details the relationship of the proposed model with previously suggested formalisms for processes modeling and querying. In particular we propose a query evaluation algorithm of polynomial data complexity that can be applied uniformly to two kind of common queries over processes, namely queries on the structure of the process specification as well as temporal queries on the potential behavior of the defined process. We show that unless $P = NP$ the efficiency of our algorithm is asymptotically optimal.

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 depends upon various business functions for support (e.g. personnel, accounting, inventory), and interacts with other BPs/activities carried out by the same or other organizations. Consequently, the implementations of such BPs typically operate 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. Complementarily, the BPEL standard (Business Process Execution Language [7]) allows description not only of the interface between the participants in a process, but also of the full operational logic of the process and its execution flow. Since BPEL has a fairly complex syntax, commercial vendors offer systems that allow design of BPEL specifications via a visual interface. These systems use a conceptual, intuitive representation of the process, as a graph of activity nodes, connected by control and data flow edges. The designs are automatically converted to BPEL specifications, which in turn can be automatically compiled into executable code implementing the BP [29]. The declarative, yet complex, nature of BPEL specifications call for the design of a query language, that will allow to effectively analyze the possible executions of a given process. To answer this need, we have developed BPQL [3,4], a query language for querying business process specifications. We have then continued to extend the query language [12–14] to account for various analysis needs that rise in the context of Business Processes, and studied query evaluation in each context. However, what is missing from these previous works is the formal, fundamental positioning of the model and query evaluation algorithm within the context of other common formalisms for modeling

* Corresponding author.
E-mail addresses: deutchd@cs.bgu.ac.il (D. Deutch), milo@cs.tau.ac.il (T. Milo).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.09.004

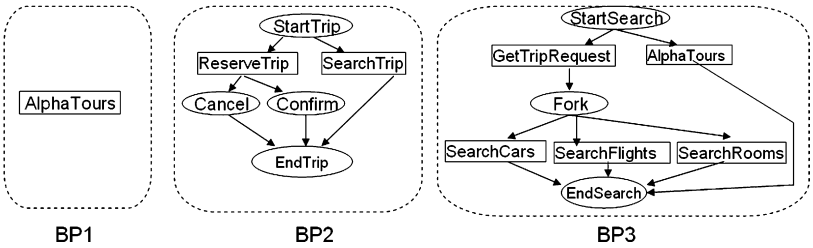

Fig. 1. A BPQL specification.

and querying processes. This positioning is the goal of the present paper. We note that some of the results presented here appeared also in [12], but only in a high-level form and without detailed proofs. Specifically, we show here that many data models are in fact equivalent (see Section 3 for a formal definition of model equivalence, and proofs that such equivalences hold) to our BP model. In particular, this means that our query evaluation results apply to these models as well. Among these commonly used models one can find restricted versions of Rewriting Systems (e.g. [32]), Recursive State Machines (RSMs) [1,6], Context Free Graph Grammars [10], and others. Each of these works relates to some query language which is evaluated over the data model. We identify two main branches of query languages, as follows. In the Databases area, the query languages are structural. Namely, they allow users to ask questions about the structure of a specification (graph). In contrast, in the verification area, the query languages are temporal [19]. Namely, the queries relate to the possible runs of the process defined by specification, and are used to identify invariants, execution patterns, etc. We provide here a unified environment for querying structural as well as temporal properties of business processes. We study the expressive power of our query language with respect to common languages, and explain how analysis tasks that cannot be expressed using temporal logic, are easily and intuitively formed using our query language. We then study the complexity of query evaluation over Business Processes, and provide query evaluation algorithms that may be applied uniformly with either the temporal or structural semantics. We show that these algorithms are practically feasible: they incur a worst case complexity that is polynomial in the size of the process specification, with the exponent dependent on the query size. We further show that the exponential dependency over the query size is unavoidable, unless $P = NP$.

Note. To guarantee a complexity that is polynomial in the size of the data, BPQL ignores the run-time semantics of certain BPEL constructs such as conditional execution and variable values, and focuses on the given specification flow. We believe that this approach offers a reasonable balance between expressibility and complexity. Clearly, the general problem is more complex, and further work is needed. The paper is organized as follows. Section 2 describes the BPQL data model and query language and its semantics. Section 3 compares BPQL to related models. Section 4 describes the query evaluation algorithm and Section 5 studies its complexity. We conclude in Section 6. Appendix A provides additional formal details on the query evaluation algorithm.

2. Preliminaries

In this section we present the formal model underlying BPQL. We start with the motivation for our work, and then proceed to the formal definitions.

2.1. Motivation

The following questions may rise from the introduction: Why are structural queries over nested graphs interesting? What are the advantages of a generic framework for multiple query semantics? Why is it important to have a graphical query language, similar to the specification? We give here intuitive answers to these questions, using some examples.

Fig. 1 depicts a partial specification of a travel agency system. The rectangle-shaped nodes represent function calls. BP1 is the root BP and contains a single node, AlphaTours, that serves as an entry point for the travel agency. BP2 describes the implementation of the AlphaTours function, where a user can choose between searching for a trip and reserving one. BP3 is the implementation of the SearchTrip function used in BP2. A user can request for a specific search (for flights, cars, etc.) or can go back to the AlphaTours trip reservation process. Note that this definition establishes recursive dependencies between the processes, as BP2 may call BP3, which in turn, if the user decides to reset (implemented in the BP as a call to AlphaTours), calls BP2.

An example query is depicted in Fig. 2. It is formulated graphically in a manner very similar to the specification. This is an important feature of the query language, as (a) it allows faster learning curve of the language and (b) it allows simultaneous formulation, by the specification designer, of a specification and verification queries over it.

To answer a query, we seek for occurrences of the described patterns within the specification. Intuitively, the query in Fig. 2 searches the AlphaTours BP, and the processes that it uses, for execution paths leading to/from a SearchFlights operation. Q2 here describes an *implementation pattern* for the AlphaTours function. The double-headed arrows indicate