Proactive Enforcement of Data Consistency by Business Processes

Xi Liu, Jianwen Su and Xuandong Li


Most of the papers available from this document appear in print, and the corresponding copyright is held by the publisher. While the papers can be used for personal use, redistribution or reprinting for commercial purposes is prohibited.
Proactive Enforcement of Data Consistency by Business Processes

Xi Liu\textsuperscript{*}, Jianwen Su\textsuperscript{‡} and Xuandong Li\textsuperscript{†}

\textsuperscript{*} State Key Laboratory for Novel Software Technology, Nanjing University, China
\textsuperscript{‡} Department of Computer Science and Technology, Nanjing University, China
\textsuperscript{†} Department of Computer Science, University of California at Santa Barbara, USA

Abstract—Data and its manipulation are essential in business processes (BPs). It is desirable to ensure within BP executions that every update to a database server guarantees to satisfy all relevant data integrity constraints (ICs). Furthermore, the earlier in a BP execution a violation is detected the more dependable the BP is. This paper studies the Process Safety Problem (PSP): will an incoming message be used in a database update causing IC violations? PSP is unsolvable in general. Taking advantage of the design-time "guard injection" technique, we propose a runtime proactive enforcement mechanism, called "process safe guarding", based on symbolic execution of BPEL processes for a bounded number of steps under "conservative strategy". Related challenges are also discussed.

I. INTRODUCTION

Data are important assets for any business to make decisions and gain global competitiveness. Modern business process modeling (BPM) languages, such as BPEL, BPMN and YAWL, are mostly data-aware. The recent artifact-centric BPM further puts data in a key role [2]. However, data management is still outside business process (BP) management, and is carried out by underlying database management systems (DBMSs). Consequently, the vital task of keeping data consistent belongs to the DBMSs. Data consistency is usually defined by specifying a set of integrity constraints (ICs) [14] that are maintained by DBMSs at runtime. It is argued recently that this "detached" method is problematic [10]. For example, when a database update fails for IC violation, it is often a challenge to locate the origin in a BP that causes the error. When data management is outsourced to some "data cloud", it is not reasonable to assume the cloud would respect application-specific ICs. It is thus desirable to have BPs responsible for producing data as well as updates adhered to ICs.

Example 1. Consider an online shopping center BPMart as Fig. 1, in which boxes are actions and the ones with cylinders at the corner denote actions that are to update (in gray) or to query (in white) underlying databases. An underlying database stores and manages the BPMart or to query (in white) underlying databases. An underlying database stores and manages the BPMart. For each item in the request, the inventory’s available quantity is updated in action “Inventory sell”: attribute avail\_qty of the instance of Inventory identified by inventory\_invId is updated to inventory.avail\_qty – order\_qty, where order and inventory are variables denoting the instances of classes (stored in database as relations) Order and Inventory, resp. Let \( \gamma \) be an IC that relates the inventory available quantity to the business revenue and product price: \( \text{avail\_qty} \geq (\text{revenue}/\text{price} \times 10\%) \) where revenue and price are also attributes of relations stored in the database. If “Inventory sell” sets avail\_qty to a “bad” value not satisfying \( \gamma \), the database integrity is destroyed. Because order\_qty and inventory\_avail\_qty can be traced back to the content of the checkout request and database, given a database (snapshot), the checkout request that results in violation of \( \gamma \) after “Inventory sell” must be blocked when it is received.

In this paper we study the Process Safety Problem (PSP): will an incoming message be used in upcoming updates in a BP that violate the ICs? This problem is generally undecidable for BPEL. The undecidability is from two directions: (a) reachability problem for BPEL programs is undecidable; (b) satisfaction problem of first-order formulas (ICs) is undecidable.

We develop the process safe guarding mechanism to proactively enforce the data consistency in BPEL processes at runtime. To avoid the undecidability of PSP, for (a), every checking only considers executions within a bounded length; for (b), we "postpone" the constraint checking until all relevant messages are received and take the "conservative" strategy assuming variables not from messages or database will cause violation of relevant ICs. Related issues, such as the correctness and applications, are also discussed.

The remainder of the paper is organized as follows. Section II compares our mechanism with related work. The process safe guarding mechanism is introduced in
Section III. Related challenges and discussions can be found in Section IV. Section V concludes the paper.

II. RELATED WORK

Data manipulation is supported in BPM languages. For example, data are part of almost every activity in BPEL. When a variable value does not conform to its XML Schema definition, faults are thrown and handled. But BPEL specification offers no way to correct the error. Besides, web service XML Schemas usually do not constraint inter-related data. Although static analysis on data dependencies in BPEL processes helps the understanding of dataflow [9], [15], and verification against data-related temporal logic properties checks the process design [5], [4], they offer no guidance to preserve data consistency in BP executions.

In the field of database theory, enforcement of ICs is not new. ICs can be statically verified before transactions [1], or checked at runtime [14]. Triggers are a powerful tool to “fix” constraint violations as a reactive means [3]. However, it is often difficult to locate the origin of the constraint violation and fix the error in BPs. It was revealed in [7] the difficulty of IC preservation in distributed databases and investigated the approach to maintain distributed ICs by reducing the necessity to look at remote databases according to specific updates. In our problem, however, concrete updates are unknown. And it is unclear that shared DBMSs would realize protocols to maintain data consistency in loosely coupled databases (e.g., [6]).

The “guard injection” mechanism is developed on rule-based workflows [10] to strengthen the enablingness condition of update actions that might violate ICs at design time. Some techniques in guard injection are used in process safe guarding. However, guard injection is not for runtime detection of violations in future executions, and therefore cannot solve PSP.

III. PROCESS SAFE GUARDING

Presented in this section is how the guarding is performed at runtime by bounded symbolic execution. First we specify the modeling of BPs with data access. Then the Process Safety Problem is formulated with the undecidable result in general case. Finally, the framework for process safe guarding is given. Note that only ideas and framework of the mechanism are discussed here. Details including algorithms and formalisms are in the full version of this paper.

A. Business processes with database access

BPEL is a widely used BPM language. Key business data are stored in DBMSs in relational model. For simplicity, we assumed that relational data is supported in BPEL.

To facilitate the checking with symbolic execution, a formal model is given for BPEL processes. The model is called BPN (Business process Petri Net), and is based on single-token colored Petri net [8] (where “single-token” means multiple tokens in one place is automatically merged).

Definition 1 (Business process Petri net (BPN)). A business process Petri net (BPN) model is a tuple,

\[ N : (P, T, \mathcal{F}, \mu_0, E, V, A, G) \]

where \((P, T, \mathcal{F}, \mu_0)\) is a single-token Petri net[13], \(E\) is the finite set of events for message input and output, \(E : T \rightarrow E\) is the event partial function on transitions, \(V\) is the finite set of variables: \(A : T ightarrow (V \times Exp(V))\) is the assignment partial functions on transitions, \(G : T ightarrow Exp(V)\) is the guarding partial function on transitions.

In Definition 1, \(A \rightarrow B\) and \(A \rightarrow B\) denote partial and total functions from set \(A\) to set \(B\) in resp., \(A \rightarrow B\) denotes total injections, and \(Exp(V)\) denotes the expressions on variables in \(V\). The behavior of the BPN model is captured by the notion of BPN runs: sequences of BPN states and transitions, where a BPN state is a tuple of marking \(\mu\) (subset of places with tokens) and variable valuation \(val\). Construction of BPN model from BPEL processes follows the approaches such as in [12], [11].

B. Process safety problem

The execution of a BPEL process \(W\) with data access is defined as a sequence of states \((\mu_i, val_i, DB_i)\) and transitions \(t_i\) (for \(i \geq 0\)). There \(\langle (\mu_0, val_0), t_0, (\mu_1, val_1), \ldots \rangle\) forms a run of the BPN model of \(W\) and \(DB_i\) is a snapshot of \(W\)'s database.

Process Safety Problem (PSP): Given a database (snapshot) that satisfies a set of ICs \(K\), a BPEL process \(W\) with

\[1\] The technical report of the full paper can be found at: http://seg.nju.edu.cn/~liux/pub/bpmandc_tr.pdf
Consider Case (2). When checking an update against some IC, update specification and IC formulas must be transformed in order to cover every possible case that affects the satisfaction of the IC. Such transformation is non-trivial, especially when an temporal database supporting instances with symbolic values is necessary in process guard. Techniques from guard injection are utilized. Moreover, all possible paths (within the bound) of the target BP must be traversed. If message variable symbols are involved when evaluating an IC, testing of the IC is skipped, and execution continues. Because the symbolic checking is called whenever a message comes, the test is actually postponed until every necessary message is received. But when all relevant messages are received, if variable symbols are still involved to evaluate an IC, these symbols denote internal variables, i.e., the variables whose value do not depend on messages or databases. Conservative strategy is taken: such execution is blocked to ensure no IC violation.

**Example 2.** As in Example 1, consider a checkout item checkoutReq.item[2]. By action “Order create” in Fig. 1, order.qty equals to checkoutReq.item[2].qty which is assumed to be 5; by action “Query Inventory”, inventory.invid equals to checkoutReq.item[2].invid which is assumed to be inv001. Suppose on the initial database snapshot, avail_qty = 9 for the Inventory identified by inv001, and (revenue/price × 10%) = 5 (see γ in Example 1). If the bound is 20 (enough to reach “Inventory sell”), symbolic checking results in FALSE for constraint γ at “Inventory sell”. Then such checkout request (checkReq) is blocked.

If the bound is 4 which is too small to reach “Inventory sell”, the violation cannot be detected within the bounded steps in symbolic execution from the reception of the checkout request. But such checking is called every 4 steps if no message is received, then the violation can be detected eventually. The update action “Inventory sell” is blocked from execution.

To illustrate the conservativeness, suppose revenue is calculated in the process rather than stored in the database or set by some message. When checking at “Inventory sell”, because internal variable revenue is involved, the checking results in FALSE no matter what value is of checkoutReq.item[2].qty.

To illustrate the postponed checking strategy, assume the response from the task pay called in “Order pay” is involved in update of available quantity. Then the checkout request is forwarded. The checking is postponed until the response from pay is received.

**IV. CHALLENGES AND DISCUSSIONS**

**Correctness and variants of process safe guarding.**

The most desired property of the process safe guarding mechanism is soundness: no IC violation throughout the execution. If there is no execution at all, no violation can occur. The notion of completeness is therefore necessary. Recall that to gain the soundness, conservative strategy is used. The notion of internally-differentiated executions is defined for a pair of executions that share the same prefix of length k, while the two k-th states are different on the value of some variable x and neither of the two (k−1)-th transitions assigns x with some message variable. A conservative execution is a sound execution with no internally-differentiated execution. The conservative completeness requires that, every conservative execution of the
original process is also an execution of the guarded process (the process with process safe guarding).

If data instances are not shared, or the DBMSs “isolates” updates from different applications, then the soundness and conservative completeness can be proved. If restrictions are made so that first order theory is decidable, conservativeness is not necessary. One example is to confine the arithmetic in BPs as: (a) integers or rational numbers with addition; or (b) real numbers with addition and multiplication.

There are some interesting directions for future work. (1) Restrict BP structure to improve the mechanism (e.g., no conservativeness) as well as to design interesting BPs. (2) Extend the mechanism to ensure soundness in shared databases with concurrent updates. (3) Incorporate mechanisms ensuring data consistency in federated databases (e.g., [6]).

Implementation and applications.

We are developing a preliminary tool to support automatic generation of the process guard checking the BP against ICs when a message is received. The tool takes a BPEL process and a file of all ICs as input, and produces the BPN model and the process guard consisting of a BPEL message filter and a Java-based BPN simulator service. The user interacts with the BPEL message filter, which in turn calls the BPN simulator that checks the bounded-step future execution for possible violations. Besides, as an application and a case study, BPMart in Fig. 1 is simplified while the complete design is in the full paper.

For further work, we consider the following issues. (1) Proper symbolic execution engine on XML data with XPath expression is necessary. (2) Interruption of process execution for checking (when the execution continues for \( k \) steps without a checking) requires to extend a BPEL engine (e.g., Apache ODE). (3) Constraint solver may be used in checking an IC. (4) Apply the mechanism and investigate the scalability and optimization on real-life complex BPs.

V. CONCLUSION

Consistency enforcement in business processes is an interesting problem and is hardly solved. This paper studies the general undecidable Process Safety Problem. Process safe guarding mechanism is suggested to solve the problem (with respect to the given bound and conservativeness), so that future execution can be sound after the proactive checking. It is expected to extend the work in several directions in the future, such as the issues in Section IV.

ACKNOWLEDGMENT

The authors would like to thank Professor Jian Yang (Macquarie University, Australia) for her informative discussion on database and business process related issues. This work is supported in part by the National Natural Science Foundation of China (No.90818022), the National Grand Fundamental Research 973 Program of China (No.2009CB320702), the National 863 High-Tech Programme of China (No.2012AA011205), and NSF grant IIS-0812578.

REFERENCES


