Data Consistency Enforcement on Business Process Transactions

Xi Liu*

State Key Laboratory for Novel Software Technology at Nanjing University
Department of Computer Science and Technology, Nanjing University, China
liux@seg.nju.edu.cn

Abstract. Transactions are common in business processes (BPs). Consistency on data, which is defined as satisfaction of a set of data integrity constraints, is one of the basic properties for business process transactions (BPTs). This requires a BPT to bring the BP execution from one consistent state to another consistent state. It is desirable to ensure within BP executions that every BPT preserves data consistency. Besides, the earlier an inconsistency is detected the less recovery is necessary. It is studied in this paper how to detect and recover from potential future inconsistency as early as possible in a BPT execution. We propose a *runtime proactive* mechanism enforcing consistency on BPTs, called "transaction consistency guarding", based on *symbolic execution* of BPEL `scope`s for bounded length and correct design of fault and compensation handlers.

1 Introduction

Data are important assets for any business to make decisions and gain global competitiveness. Transactions are common in business processes (BPs). *Consistency on data* is one of the most desired properties for business process transactions (BPTs). That is, business data must adhere to a set of integrity constraints (ICs) [14] both when the transaction just starts and when the transaction just completes.

In most current business process models, data management is outside BP management, and is carried out by underlying database management systems (DBMSs). Consequently, the vital task of keeping data consistent belongs to the DBMSs. However, this "detached" method is problematic, as argued in [11]. Besides, compared to database transactions [14], BPTs often involve interactions and collaboration with parties and applications outside the BP, and can last for a long duration. Transactional mechanisms in DBMSs, such as roll-back, locking, etc. [14], do not work for BPTs [6,2]. Compensation is used instead for recovery from failure. It is therefore desirable to have BPs responsible for enforcing data consistency on BPTs.

Example 1. Consider an online shopping center BPMart (partially shown in Fig. 1) whose data are managed in an underlying database. Assume IC γ relates the inventory available quantity to the business revenue and product price; defined as $\gamma ::= avail\_qty \geq (revenue/price \times 10\%)$, where *revenue* and *price* are also attributes of relations stored

* Supported by the National Grand Fundamental Research 973 Program of China (No.2009CB320702).

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 394–399, 2013.

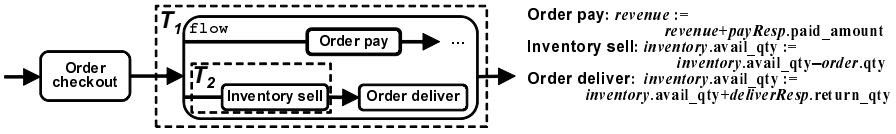

Fig. 1. A segment in BPMart, where round-corner boxes denote actions (details not included), dotted boxed denotes BPEL scopes (fault and compensation handler not included). Updates to the database by the actions correspond to calculations listed on the right.

in the database. The available quantity is updated in "Inventory sell" in BPT T_2: attribute *avail_qty* of the instance of *Inventory* corresponding to *inventory* is updated to $inventory.avail\_qty - order.qty$, where *order* and *inventory* are variables denoting the instances of business data classes (stored in database as relations) *Order* and *Inventory*, resp. If "Inventory sell" sets *avail_qty* to a "bad" value not satisfying γ, the database integrity is destroyed when T_2 commits. Since the calculation in "Inventory sell" is deterministic, with the value of *inventory.avail_qty*, *revenue* and *price*, such inconsistency can be detected as soon as T_2 starts. Thus, T_2 aborts immediately and prevents inconsistency caused by the update in "Inventory sell".

Preservation of data consistency by BPs was studied in [11], in which the "guard injection" technique is developed on rule-based workflows. The updates that might violate some ICs are blocked before they are about to execute. No transactional structure is considered in guard injection.

It is preferred to detect potential data inconsistency as early as possible in a BPT execution, so that the BPT aborts immediately and the compensation would be simpler in order to recover the consistency. BPTs are defined using `scopes` in BPEL. So we look into the Transaction Consistency Problem (TCP): will data inconsistency be detected when a BPEL scope starts? Unfortunately, TCP is generally undecidable.

We developed the *transaction consistency guarding* mechanism to *proactively* enforce consistency of BPEL scopes at runtime. To tackle the undecidability of TCP, only scope executions within bounded length is considered, and strategies like postponed checking and conservativeness are used to check IC satisfaction until all relevant messages are received and assume variables not from messages will cause IC violation.

The proposed transaction consistency guarding mechanism symbolically executes the BPEL scopes for bounded number of steps. When the commitment point of the scope is reached within the bounded length, and inconsistency is detected, the BPEL scope aborts (as if a fault occurs). The scope's fault handler is triggered to recovery consistency, in which sub-scopes are compensated to reverse the committed effect. This mechanism requires correct design of BPTs with fault handling and compensation (such as BPEL scopes). Issues concerning correctness of the proposed mechanism are also discussed: correct design of BPTs as well as soundness and conservative completeness.

The remainder of the paper is organized as follows. Section 2 compares our mechanism with related work. The transaction consistency guarding is introduced in Section 3. Correctness issues are discussed in Section 4. Section 5 concludes the paper.

2 Related Work

Data manipulation and compensable transactions are supported in most BP models. In BPEL, when a variable value does not conform to its definition, faults are thrown and handled. However, current SOA specifications and BP models (e.g., BPEL, BPMN, YAWL, WS-Coordination) put no requirement on effect of transactions. Thus there is no way to ensure data consistency. 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 [10,16], and verification against data-related temporal logic properties checks the process design [5,4], they offer no guidance for data consistency on BPTs.

In the field of database theory, enforcement of ICs is not new. ICs can be statically verified before short-lived database 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 [9] 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., [8]).

Compensation is introduced in Sagas [6]. Although compensation is supported in most BP models, consistency cannot be automatically recovered [7]. In contrast with requiring nothing on the compensation, theoretical work such as cCSP requires the whole world must be restored [2]. This can help to recover consistency. However, due to the open-world assumption of BPTs, such requirement is hardly practical. It is also possible to extend the DBMSs to manage BPTs by adding new intermediate layers, e.g. [15]. But this introduces more complexity and has no feedback to BPs.

The "guard injection" mechanism is developed on rule-based workflows [11] to strengthen the enableness condition of update actions that might violate ICs at design time. However, guard injection is not for runtime detection of violations in future executions of transactions, and therefore do not solve TCP.

3 Transaction Consistency Guarding

Presented in this section is how the guarding is performed at runtime. First the Transaction Consistency Problem is formulated with the undecidable result in general case. The framework for transaction consistency guarding is given next.

3.1 Transaction Consistency Problem

BPEL is a widely used BP modeling language supporting compensable BPT as scopes. Key business data are stored in DBMSs in relational model. For simplicity, we assumed that relational data is supported in BPEL. The semantics of BPEL can be modeled by Petri net [12] or process algebra [13]. The execution of a BPEL process with data access is defined, in this paper, as an sequence of states (s_i, DB_i), denoting the BPEL process

state s_i (with variable valuation) together with database snapshot DB_i, and transitions t_i, denoting firing a basic process step (such as a basic activity), where $i \geq 0$.

Transaction Consistency Problem (TCP): *Given a database (snapshot) satisfing a set of ICs* **K**, *a BPEL process* $\mathcal{W}$ *with a scope-based transition T, when the execution of T starts in* $\mathcal{W}$, *will it terminates in a state whose database snapshot violates* **K***?*

Theorem 1. *Transaction Consistency Problem is undecidable.*

Theorem 1 implies unsolvability to force abortion at the beginning of exactly the set of "bad" BPTs. The undecidability comes from the fact that general BPEL processes (and its scopes) are Turing-complete; and satisfaction problem of first-order formulas, that is the integrity constraints, are undecidable. However, we take strategies like bounding the length of execution for checking and the "conservativeness", and develop a "sound" solution for TCP.

3.2 Transaction Consistency Guarding Framework

The runtime guarding is performed by checking in symbolic execution of the BPEL scopes. When violation is detected, fault handler of the current scope is triggered to restore the IC related data so that the data consistency is recovered.

In the symbolic execution, the actual database should not be updated. A simulation database supporting instances with symbolic values is assumed. Define the database for symbolic execution be the union of the actual database and the symbolic database.

The mechanism is performed by a "process guard" as an extension to BPEL engines. Let a positive integer k be the given bound. The process guard monitors the execution of the target process $\mathcal{W}$ and symbolically executes the scope to check violation for the next k steps. The checking starts whenever: (1) a scope starts; (2) a message is received in a scope's execution; or (3) the execution of a scope has continued k steps or more without a checking. Let the scope in question be denoted T. The checking is performed by symbolic execution from current state s to detect inconsistency in the next k steps. If commitment of T is reached within the k-step symbolic execution, and some IC is violated, T is forced to abort at the state s, i.e., the process guard raise a fault on state s. Running sub-scopes are also aborted; then the parent scope's abortion waits until the fault handling of the sub-scopes finishes. Otherwise, the execution continues.

All possible executions (within the bound) must be traversed. When message variable symbols are involved in symbolically checking of some IC, the checking is skipped and execution continues. Because the symbolic checking is called whenever a message comes, the test is actually "postponed" until all necessary messages are received. But if variable symbols are still involved to evaluate the constraint when all relevant messages are received, these symbols denote internal variables (i.e., the variables whose value do not depend on messages). "Conservative" strategy is taken: such scope aborts.

Example 2. Consider the cases as in Example 1. Suppose when T_2 starts, $avail\_qty = 9$ for the *Inventory* instance corresponding to variable *inventory*, $order.qty = 5$ and $(revenue/price \times 10\%) = 5$. If the bound is enough to finish T_2, inconsistency on γ (see Example 1) is detected in the symbolic execution. Then a fault is raised in T_2. Because no actual update is yet made, consistency is preserved.

Consider scope T_1 in Fig. 1, in which business revenue is updated by the actual paid amount in "Order pay"; and, action "Order deliver" replenishes the inventory with returned products. Assume when T_1 starts, $avail\_qty = 9$ for the *Inventory* instance corresponding to variable *inventory* and $order.qty = 5$. Suppose T_2 starts before "Order pay", so that $(revenue/price \times 10\%) = 4$ when T_2 starts. Then T_2 completes successfully. The satisfaction of γ on T_1 depends on both messages *paidResp* and *deliverResp*. The checking is postponed until both messages are received. If then, γ fails to hold, a fault is raised in T_1. Provided that the fault handler of T_1 with compensation of T_2 can reverse the calculation made on $inventory.avail\_qty$, *revenue* and $order.qty$, the consistency on γ is recovered.

4 Correctness of Transaction Consistency Guarding

Correctness of BPTs and Compensation
Transaction consistency guarding mechanism relies on the fault and compensation handler to restore data consistency. Because BPEL relies completely on designers to write the code for fault handling and compensation, and ideal recovery (such as [2]) is impractical, correct design of scopes must be investigated.

There are several issues concerning the compensation correctness. First, certain variables must be selected to be critical to scopes. Such variables affect the consistency on ICs and updates to the databases. Second, correctness criteria for scopes must be defined. Such criteria should require the fault and compensation handler of scopes to restore the value of critical variables to the same value as the scope (instance) starts. Note that the fault can also be raised by the process guard when inconsistency is detected in symbolic execution of the scopes. Third, verification techniques need to be studied to ensure that the scope's design conforms to the correctness criteria. Such verification may be done by theorem proving, program analysis and (bounded) model checking.

Correctness and Variants of Transaction Consistency Guarding
The most desired property of the process safe guarding mechanism is *soundness*: no IC violation throughout the execution. However, it is the fact that, when there is no execution at all, no violation can occur. The notion of completeness is therefore necessary. Recall that to gain soundness, conservative strategy is used. The *conservative completeness* requires that, every execution of the original scope without IC violation or assignment to internal variable (variables do not depend on messages) is also an execution of the guarded process. Provided that the fault and compensation handler of every scope can reverse the changes made on IC-related variables in the scope activity, the consistency guarding mechanism is both sound and conservative complete.

Conservativeness may cause scope abortion when no inconsistency occur in actual executions. However, this is necessary to ensure soundness. If restrictions are made so that first order theory is decidable, conservativeness is not necessary. One example is to confine the arithmetic in scopes as: (a) integers or rational numbers with addition; or (b) real numbers with addition and multiplication. Confining the scopes' structure can also help the verification of scopes against desired transactional properties [4].

5 Conclusion

Consistency enforcement in business process transactions is an interesting problem but is hardly solved. To find future inconsistency as early as possible, we first study the generally undecidable Transaction Consistency Problem. Transaction consistency guarding mechanism is suggested with respect to the given bound and strategies of postponed checking and conservativeness, so that after the proactive checking and recovery by the fault handling and compensation, consistency of the transaction is ensured. The correctness of the mechanism depends on correct design of compensable transactions. The transactions may also be restricted to allow relaxed mechanism (e.g., no conservativeness) and to help the verification of compensation design. Future work can be on issues such as: realization and application of the mechanism, validation on real-life BPs, transaction design assurance and more interesting restriction on transaction structures.

References

1. Benedikt, M., Griffin, T., Libkin, L.: Verifiable properties of database transactions. In: Proc. of Symposium on Principles of Database Systems (PODS), pp. 117–127 (1996)
2. Butler, M., Hoare, S.T., Ferreira, C.: A Trace Semantics for Long-Running Transactions. In: Abdallah, A.E., Jones, C.B., Sanders, J.W. (eds.) CSP 25. LNCS, vol. 3525, pp. 133–150. Springer, Heidelberg (2005)
3. Ceri, S., Widom, J.: Deriving production rules for constraint maintainance. In: Proc. Int. Conf. on Very Large Data Bases (VLDB), pp. 566–577 (1990)
4. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. of Int. Conf. on Database Theory (ICDT), pp. 252–267 (2009)
5. Fu, X., Bultan, T., Su, J.: Model checking XML manipulating software. In: Proc. Int. symposium on Software Testing and Analysis (ISSTA), pp. 252–262 (2004)
6. Garcia-Molina, H., Salem, K.: Sagas. In: Proc. of Int. Conf. on Management of data (SIGMOD), pp. 249–259 (1987)
7. Greenfield, P., Fekete, A., Jang, J., Kuo, D.: Compensation is not enough. In: Proc. of Int. Conf. on Enterprise Distributed Object Computing (EDOC), pp. 232–239 (2003)
8. Grefen, P., Widom, J.: Protocols for integrity constraint checking in federated databases. Distrib. Parallel Databases 5, 327–355 (1997)
9. Gupta, A., Widom, J.: Local verification of global integrity constraints in distributed databases. In: Proc. of Int. Conf. on Management of Data (SIGMOD), pp. 49–58 (1993)
10. Kopp, O., Khalaf, R., Leymann, F.: Deriving explicit data links in WS-BPEL processes. In: Proc. of Int. Conf. on Services Computing (SCC), pp. 367–376 (2008)
11. Liu, X., Su, J., Yang, J.: Preservation of integrity constraints by workflow. In: Proc. of Int. Conf. on Cooperative Information Systems (CoopIS), pp. 64–81 (2011)
12. Lohmann, N.: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008)
13. Qiu, Z., Wang, S., Pu, G., Zhao, X.: Semantics of BPEL4WS-like fault and compensation handling. In: Proc. of Int. Conf. on Formal Methods, pp. 350–365 (2005)
14. Ramakrishnan, R., Gehrke, J.: Database Management Systems, 3rd edn. McGraw-Hill (2002)
15. Wang, R., Salzberg, B., Lomet, D.: Log-based middleware server recovery with transaction support. The VLDB Journal, 1–24 (2010)
16. Amme, W., Martens, A., Moser, S.: Advanced verification of distributed WS-BPEL business processes incorporating CSSA-based data flow analysis. Int. Journal of Business Process Integration and Management 4(1), 47–59 (2009)