Seven Challenges for RESTful Transaction Models

Nandana Mihindukulasooriya, Miguel Esteban-Gutiérrez, Raúl García-Castro
Center for Open Middleware, Ontology Engineering Group
Universidad Politécnica de Madrid, Spain
{nmihindu,mesteban,rgarcia}@fi.upm.es

ABSTRACT

The REpresentational State Transfer (REST) architectural style describes the design principles that made the World Wide Web scalable and the same principles can be applied in enterprise context to do loosely coupled and scalable application integration. In recent years, RESTful services are gaining traction in the industry and are commonly used as a simpler alternative to SOAP Web Services.

However, one of the main drawbacks of RESTful services is the lack of standard mechanisms to support advanced quality-of-service requirements that are common to enterprises. Transaction processing is one of the essential features of enterprise information systems and several transaction models have been proposed in the past years to fulfill the gap of transaction processing in RESTful services. The goal of this paper is to analyze the state-of-the-art RESTful transaction models and identify the current challenges.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; H.1 [**Information Systems**]: Models and Principles; H.2.4 [**Information Systems**]: Transaction processing

Keywords

REST, Transactions, Challenges

1. INTRODUCTION

REpresentational State Transfer (REST) architectural style, initially known as "HTTP object model", was developed as a means of communicating Web concepts and is the foundation for the modern Web architecture [4]. REST introduces several architectural constraints on hypermedia systems design such as *resource identification* (addressability), *uniform interface*, *stateless interactions*, *self-describing messages*, and *hypermedia as the engine of application state* (HATEOAS). These constraints induce certain desirable properties that enable the development of loosely coupled scalable systems.

However, not every web application (including those calling themselves *RESTful*) adheres to all these REST constraints. Models such as the *Richardson Maturity Model* categorize services according to their adherence to the REST constraints [24], and provide an insight about the impact and consequences of dropping these constraints.

Nonetheless, Web Services built following the REST architectural constraints (*RESTful services*, from now on) are getting traction in the industry in recent years as a simpler alternative to application integration. However, one of the main criticisms is the lack of standard mechanisms to support the advanced quality of service requirements that are required by enterprises [14].

Transaction support is an important quality-of-service requirement in most enterprise business scenarios. A real life transaction example would be transferring money from one account to another in a banking application. Both the deduction of money from one account and the addition to the other should happen in an "all-or-nothing" manner and the intermediate inconsistent states such as when only one account is modified should not be visible outside the transaction. In computer applications, a transaction is defined as a sequence of operations on the physical or abstract application state that can be considered as a single unit of work [7]. Gray defined the transaction concept with *atomicity*, *durability*, and *consistency* [6] and Haerder & Reuter coined the acronym *ACID* adding *isolation* to the aforementioned three properties [8].

Beyond this basic *flat* transaction definition, further transaction types have been developed in order to meet the requirements of other real-life complex transactional scenarios: *chained transactions*, *nested transactions*, *distributed transactions*, *long-lived transactions*, etc [7].

However, the strong consistency property of the ACID model may hinder other quality aspects of data-sharing systems. According to the *CAP* theorem [2], these systems can only exhibit at most two of the following three properties: *consistency*, *availability*, and *tolerance to network partitions*. Furthermore, even in the absence of network partitions, data replication based high-availability systems require a tradeoff between *consistency* and *latency* as stated by the *PACELC* theorem [1]. To overcome these issues, other consistency models propose to make a compromise between *consistency* and *availability/latency* by relaxing consistency guarantees in order to catter for network partition fault-tolerance and high-availability (see *eventual consistency* [23] and *BASE* [15]).

Up to now, there have been several efforts to define ACID-based *flat* transactions for RESTful services. In this paper we analyze the state-of-the-art to this extent and identify the current challenges for REST-compliant strongly consistent transaction models.

The rest of the paper is organized as follows: Section 2 provides an overview of RESTful transactions with different characteristics and analysis of existing models; Section 3 presents a set of challenges that were identified based on the previous analysis; and, Section 4 draws some conclusions.

2. RESTFUL TRANSACTIONS

2.1 RESTful transaction characteristics

In order to understand the nature of transactional scenarios for RESTful applications, we have characterized them according to three dimensions: (1) the *resources* that are involved in the trans-

Copyright is held by the author/owner(s).
WWW'14 Companion,, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2579218.

action; (2) the *workflow of actions* that will be carried out during the transaction; and (3) the specific characteristic of the *business application domain*.

Resource characteristics include the number of resources (cardinality), the ownership and management of resources, and their physical distribution. The characteristics of the *workflow* dimension include flexibility (i.e., whether the transaction workflow has to be known at design time or it can be defined on-the-fly at runtime), whether the actions are interactive or not, whether the actions are organized as a flat sequential workflow or include more complex organizations such as chains or hierarchies, and whether the participating resources have to have a pre-agreement (context) or they are loosely-coupled (i.e. any resource in the wild can participate in a transaction). Finally, *domain-specific* characteristics include the average expected duration of a transaction, the reversibility or compensability of actions, and the level of transactional guarantees required by the business use case. Table 1 summarizes the different characteristics.

Dimension	**Characteristic**	**Variations**
Resource	Cardinality	Single
		Multiple
	Management	Centralized authority
		Decentralized authority
	Distribution	Single node
		Distributed
Workflow	Flexibility	Predefined
		Free-form
	Interactivity	Interactive
		Non-interactive
	Structure	Flat
		Chained
		Hierarchical
	Scope	Predefined context
		Global
Domain-specific	Lifetime	Short-lived
		Long-lived
	Actions	Reversible
		Nonreversible
		Compensable
	Required guarantees	ACID
		BASE
		Reservation (ACD)

Table 1: Characteristics of RESTful transaction scenarios

2.2 RESTful transaction models

The most intuitive way to support transactions in RESTful applications is to design the resource model in a way that state transitions that have to happen in a transactional manner can be done through a single resource by introducing coarse-grained resources that capture the complete transactional state (e.g. two account resources vs. a single money transfer resource). However, looking at the different characteristics identified in the previous section, it becomes clear that this is not always possible. Thus, several transaction models have been proposed for RESTful services over the past years as summarized in Table 2. In addition to the aforementioned approaches, there are other implementation oriented approaches such as REST-* / JBoss JAX-RS transaction support (both ACID and compensating transactions)[1].

We have analyzed the aforementioned models using an example scenario of updating two resources with each model, summarized in Table 3. The goal was not to select one model as the best model but to understand the state of the art about RESTful transaction models. First we looked at the model's ability to provide ACID guarantees in transactions. One thing to note is that while atomicity and isolation are guaranteed by the transaction protocols, consistency validation and durability are mostly guaranteed by the implementation (thus are not included in Table 3). Another criterion was the RESTfulness of the model, remarking whether or not the protocol communications adhere to relevant REST constraints.

[1] `https://community.jboss.org/wiki/TransactionalSupportForJAXRSBasedApplications`

Key	**Year**	**Transaction Model**
1	∼2000	Batched transactions with overloaded POST
2	2007	Transactions as resources [18]
3	2009	Optimistic technique for transactions using REST [19]
4	2009	A consistent and recoverable RESTful transaction model (RETRO) [12, 17, 16]
5	2010	Timestamp-based two phase commit protocol for RESTful services (TS2PC4RS) [20, 22, 21]
6	2011	Try-Cancel/Confirm pattern (TCC) [13]
7	2012	Atomic REST batched transactions [9]

Table 2: RESTful transaction models

Property	**Transaction models**						
	1	2	3	4	5	6	7
Transaction properties							
Atomicity	✓	✓	✓[1]	✓	✓	✓	✓
Isolation	✓	✓[2]	X	✓	X	X	✓
REST Constraints							
Uniform interfaces	✓	X	✓	✓	✓	✓	✓
Statelessness	✓	✓[3]	X	✓[3]	X	✓	✓
HATEOAS	X	X	X	✓	X	✓	X
HTTP related properties							
Semantics not violated	✓	✓	✓	✓	X	✓	✓
Common verbs supported	✓	✓	X	X	✓	✓	✓
Low overhead	✓	✓	✓	X	X	✓	✓
Miscellaneous properties							
Optionality	✓	?	?	✓	?	?	✓
Discoverable	?	?	?	✓	?	?	✓
Distributed transactions	X	X	✓	?	✓	✓	?
Theoretical proofs	?	?	?	✓	✓	✓	?
Implementation available	✓	?	?	✓	?	✓	✓
Performance evaluation	?	?	?	?	?	✓	?
Legend - ✓True / X False / ? Unknown or not defined in the model							
1 - Given the actions can be compensated 2 - Possible lost update problem 3 - See section 3.3							

Table 3: Analysis of existing RESTful transaction models

Because in practice most RESTful services are implemented using HTTP, we verified that the models do not violate HTTP semantics (e.g. safety and idempotency of certain operations), support the commonly used HTTP verbs (GET, PUT, POST, and DELETE) and only use the standard verbs defined in HTTP/1.1. These aspects are important for the interoperability and wide adoption of the model. In addition, we analyzed the overhead added by the transaction protocol (in the success case) to the communication, i.e. additional HTTP round trips and payload data.

Further, other miscellaneous properties were considered: (a) *protocol optionality*, that is, whether or not servers and clients that do not support the protocol can co-exist with others that support it (this should facilitate the progressive adoption of the model); (b) *discoverability*, that is, all the metadata needed to execute the transactions can be discoveredws in a RESTful manner without out-of-band knowledge (i.e. following links); (c) the availability of theoretical proofs that demonstrate whether or not the model is correct; (d) the availability of implementations; and (e) the provision of an evaluation of the overhead introduced by the protocol.

The results of the analysis show that most models fail to fulfill several desirable properties and cannot be used in some of the scenarios identified in Table 1 due to some challenges for RESTful transaction processing that will be discussed in the next section.

3. CHALLENGES FOR RESTFUL TRANSACTIONS

Based on the analysis of the existing transaction models and comparing them with the different characteristics of RESTful transaction scenarios, we have identified the following challenges.

3.1 Decentralized authorities

Transactions that involve resources managed by multiple authorities is one of the main challenges in current RESTful transaction models. The main problem of decentralized authorities is the need for coordination and agreement with regards to the final outcome of a transaction whilst ensuring its atomicity, an issue that requires complex failure modes and recovery mechanisms [3]. This is a common problem in distributed computing that is typically solved using a consensus protocol, i.e., the two-phase commit (usually the XA protocol). However, the majority of the RESTful transaction models do not cover this scenario and the challenge is the design a stateful consensus protocol without violating REST constraints.

3.2 Distributed servers

Distributed systems in which the ordering and timing of events is relevant[2] require the synchronization of *logical* clocks of different nodes [10]. Currently, mechanisms such as Lamport timestamps and vector clocks are used for ordering events in distributed systems [10]. How these approaches can be applied in REST services for ordering the actions on different resources and how timestamps can be used consistently still remains a challenge to be solved.

3.3 Statelessness and isolation

The *statelessness* REST constraint states that servers should be stateless and should not maintain any conversation state with the client (client-stateless-server) [5]. However, the *isolation* ACID property states that any intermediate change of a transaction should not be visible to ongoing parallel transactions. This requires servers to maintain intermediate states for actions that are not committed by maintaining a session state for a transaction. Thus, these two properties are in conflict.

Current isolation-preserving REST transaction models solve this problem representing these session states as a set of temporary resources that have their own identifiers (URLs). Despite this approach aligns with W3C best practices[3], it is arguably a REST anti-pattern, as those temporary resources do not represent resource state but application (or session) state. Furthermore, this approach introduces a new challenge: link transparency. When working with temporary resources, it is necessary to distinguish links that point to temporary resources from those that point to original resources, so that when the transaction is committed, all the links of original representations point to original resources.

An alternative approach to solve this issue could be the usage of a mechanism similar to that proposed by the Memento framework[4] for providing access to representations of different resource states using the same identifier (URL). However, this approach directly violates the stateless REST constraint.

[2]Those in which agents residing in different nodes of the system have to perform actions in a particular order.

[3]`http://www.w3.org/2001/tag/doc/IdentifyingApplicationState#UseURIsforStates`

[4]`http://www.ietf.org/rfc/rfc7089.txt`

3.4 Availability, deadlocks, and fairness guarantees

Locking has been the prominent solution for achieving isolation in transactions in the database field [7] and most RESTful transaction models have followed the same path. However, there are several issues that need to be taken care of when using this technique, in particular: availability, deadlock prevention, and fairness guarantees.

Availability is a fundamental aspect of distributed applications, therefore transaction models should minimize the negative effects of locks on the availability of resources. This issue is deepened by the fact that is REST applications operations take longer due to transport overheads (HTTP).

Deadlocks and resource starvation are common problems when locks are not used consistently or when fairness is not guaranteed. These become important specially when the acquisition and release of locks is managed by different clients. One corner case would be a misbehaved client (or a client with a defect) not releasing the locks after it has finished with a transaction.

Current approaches use two-phase locking with a growing phase and a shrinking phase to prevent deadlocks, and use timeouts to get some degree of fairness (lock auto-release after timeout). However, the enforcement of two-phase locking and achieving fairness remains a challenge for the RESTful transaction models.

Another alternative is to use optimistic concurrency control mechanisms provided by HTTP using conditional updates with ETags. However, this approach does not guarantee isolation as intermediate states of the resources become visible outside the transaction.

3.5 Resource granularity and composition

REST allows resources to be at different granularity levels. Thinking in a hierarchical model, an application could simultaneously provide a high-level view of an entity via a coarse-grained resource and a detailed view using a fine-grained resource. Also, *collection* resources found in specifications such as AtomPub[5], Hydra [11], or the Linked Data Platform[6] are special cases of resource composition. These particular cases lead to problematic situations when locks are used with these resources, i.e., locking a specific resource might not prevent the information carried in that resource from being read or updated because this information is not exclusively bounded to such resource. Thus, resource locking might not effectively prevent the access to the locked resource state since the same data may be exposed by a different resource that is not being locked. Managing the overall consistency when the same state is exposed via multiple resources remains a challenge for RESTful transaction models.

3.6 Heuristic generation

Most of the transaction models make use of heuristics when deciding on certain transaction parameters such as the timeouts used in [12, 13]. In this case, generating a suitable timeout is a challenge because it not only affects the performance but the correctness of the model, i.e., a premature timeout can decrease the performance or make the system consistently fail [13]. In scenarios that involve decentralization and distribution, heuristics generation is even more difficult since most of the information is (a) not known in advance, and (b) not known by a single party. Most of the RESTful transaction models do not provide algorithms nor guidelines for heuristic generation, and thus remains as a challenge.

[5]`http://atompub.org/`

[6]`http://www.w3.org/TR/ldp/`

3.7 Gap between research and industry

Though several transaction models have been proposed in the past decade, only few are used in industry. Out of the current approaches, the overloaded POST method seems to be the most widely used mechanism for REST transactions due to its simplicity and efficiency. However, it has a main disadvantage: it cannot handle distributed and decentralized authority scenarios.

It is worth taking a look at why the other approaches are not taking as much traction. One of the key issues is the complexity and overhead added by the transaction mechanisms. Another aspect is that they are defined on their own, when in practice they have to be integrated with existing development frameworks as well as to take into account other cross-cutting concerns, i.e., security. Thus, the challenge is defining a simple yet efficient REST-compliant protocol that provides transactional guarantees, which can be seamlessly integrated with other technologies of the REST development stack.

4. CONCLUSIONS

The main conclusion of the analysis of the existing RESTful transaction models is that *one model does not fit all*. RESTful transaction scenarios are diverse in many dimensions and no transaction model fulfills the requirements of every scenario. On the contrary, these models are designed to cover specific scenarios. However, there are still some scenarios that are not sufficiently supported by the current models.

In this paper we have identified several challenges that have been overlooked by current models, which have to be considered when addressing the uncovered RESTful transaction scenarios. Some of the challenges are similar to those faced by distributed database transactions (i.e., decentralized authorities and distributed servers) while others (i.e., statelessness and resource granularity) are specific to REST architectural style. Thus, it is worth to take a look at how these problems are solved in database and distributed systems research areas to evaluate whether the same solutions apply or how they can be adapted in the context of RESTful services.

5. ACKNOWLEDGMENTS

This research is supported by the ALM iStack project of the Center for Open Middleware.

6. REFERENCES

[1] Abadi, D.J.: Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. Computer 45(2), 37–42 (2012)

[2] Brewer, E.A.: Towards Robust Distributed Systems. In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing. p. 7. PODC '00, ACM, New York, NY, USA (2000)

[3] Coulouris, G., Dollimore, J., Kindberg, T., Blair, G.: Distributed Systems: Concepts and Design, 5th edition. Addison-Wesley (2011)

[4] Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT) 2(2), 115–150 (2002)

[5] Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California (2000)

[6] Gray, J.: The Transaction Concept: Virtues and Limitations (Invited Paper). In: Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7. pp. 144–154. VLDB '81, VLDB Endowment (1981)

[7] Gray, J., Reuter, A.: Transaction processing. Kaufmann (1993)

[8] Haerder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR) 15(4), 287–317 (1983)

[9] Kochman, S., Wojciechowski, P.T., Kmieciak, M.: Batched transactions for RESTful web services. In: Current Trends in Web Engineering, pp. 86–98. Springer (2012)

[10] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978)

[11] Lanthaler, M., Guetl, C.: Hydra: A Vocabulary for Hypermedia-Driven Web APIs. In: Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M., Auer, S. (eds.) LDOW. CEUR Workshop Proceedings, vol. 996. CEUR-WS.org (2013)

[12] Marinos, A., Razavi, A., Moschoyiannis, S., Krause, P.: RETRO: A consistent and recoverable RESTful transaction model. In: Web Services, 2009. ICWS 2009. IEEE International Conference on. pp. 181–188. IEEE (2009)

[13] Pardon, G., Pautasso, C.: Towards distributed atomic transactions over RESTful services. In: REST: From Research to Practice, pp. 507–524. Springer (2011)

[14] Pautasso, C., Zimmermann, O., Leymann, F.: Restful Web Services vs. "Big"' Web Services: Making the Right Architectural Decision. In: Proceedings of the 17th International Conference on World Wide Web. pp. 805–814. WWW '08, ACM, New York, NY, USA (2008)

[15] Pritchett, D.: BASE: An Acid Alternative. Queue 6(3), 48–55 (May 2008)

[16] Razavi, A., Marinos, A., Moschoyiannis, S., Krause, P.: Recovery management in RESTful interactions. In: Digital Ecosystems and Technologies, 2009. DEST'09. 3rd IEEE International Conference on. pp. 419–424. IEEE (2009)

[17] Razavi, A., Marinos, A., Moschoyiannis, S., Krause, P.: RESTful transactions supported by the isolation theorems. In: Web Engineering, pp. 394–409. Springer (2009)

[18] Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly (2008)

[19] da Silva Maciel, L.A.H., Hirata, C.M.: An optimistic technique for transactions control using REST architectural style. In: Proceedings of the 2009 ACM symposium on Applied Computing. pp. 664–669. ACM (2009)

[20] da Silva Maciel, L.A.H., Hirata, C.M.: A timestamp-based two phase commit protocol for web services using rest architectural style. Journal of Web Engineering 9(3), 266–282 (2010)

[21] da Silva Maciel, L.A.H., Hirata, C.M.: Extending timestamp-based two phase commit protocol for RESTful services to meet business rules. In: Proceedings of the 2011 ACM Symposium on Applied Computing. pp. 778–785. ACM (2011)

[22] da Silva Maciel, L.A.H., Hirata, C.M.: Fault-tolerant timestamp-based two-phase commit protocol for RESTful services. Software: Practice and Experience (2012)

[23] Vogels, W.: Eventually Consistent. Queue 6(6), 14–19 (Oct 2008)

[24] Wilde, E., Pautasso, C.: REST: From Research to Practice. Springer (2011)