The Atomic Web Browser

Cesare Pautasso
Faculty of Informatics
University of Lugano, Switzerland
c.pautasso@ieee.org

Masiar Babazadeh
Faculty of Informatics
University of Lugano, Switzerland
masiar.babazadeh@usi.ch

ABSTRACT
The Atomic Web Browser achieves atomicity for distributed transactions across multiple RESTful APIs. Assuming that the participant APIs feature support for the Try-Confirm/Cancel pattern, the user may navigate with the Atomic Web Browser among multiple Web sites to perform local resource state transitions (e.g., reservations or bookings). Once the user indicates that the navigation has successfully completed, the Atomic Web browser takes care of confirming the local transitions to achieve the atomicity of the global transaction.

Categories and Subject Descriptors
H.4.3 [**Information Systems**]: Information Systems Applications

Keywords
Atomic Transactions, Distributed Systems, REST

1. INTRODUCTION
The REST architectural style [1] features excellent support for the reliable interaction of clients with a single resource. Considering that `GET`, `PUT`, `DELETE` requests are by definition idempotent, any failure during these interaction can be addressed by simply repeating the request. However, no guarantees can be made for more complex interactions which require to atomically transfer state among resources distributed across multiple RESTful Web services [6]. For example, when a Web browser interacts with more than one e-commerce service, we want to ensure that all interactions can be performed atomically in order to complete the reservation of all the purchased goods as a single step.

The goal of this poster is to present a browser extension that solves the problem of achieving atomicity within distributed RESTful APIs within the constraints of: 1) Using a lightweight transaction model (e.g., ATOMIKOS TCC [4]); 2) Avoiding changes to the HTTP protocol; 3) Keeping the services unaware that they are part of the transaction; 4) Focusing on the atomicity property of transactions.

The problem about how to transparently deal against failure scenarios within workflows spanning multiple RESTful services is an important one. Our solution provides the ability to group multiple RESTful interactions and treat them as a single logical step, as well as to ensure that it is possible to guarantee the consistency of a set of resources which are distributed over multiple servers.

2. PROTOCOL
1) The Atomic Web Browser goes about interacting with multiple RESTful service APIs.

2) Interactions may lead to a state transition of the service identified by some URI. We assume that this URI can be later used to confirm (or cancel) it within a given time.

3a) Once the workflow successfully completes, the set of confirmation URIs is used to confirm the state transitions of the services with idempotent requests (e.g., `PUT`).

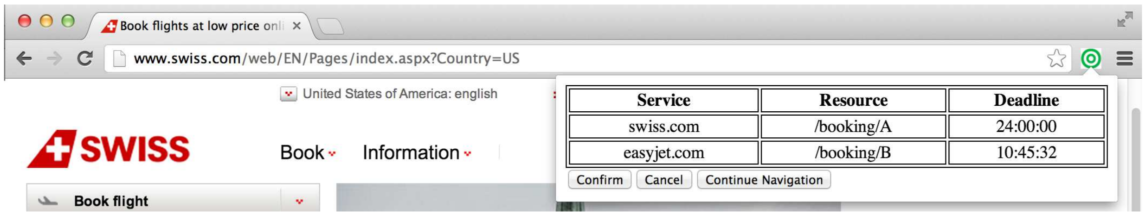
3b) If the workflow fails, the set of cancellation URIs that have been collected until the failure occurs are used to signal each of the services with idempotent messages (e.g., `DELETE`) to cancel their state transition.

Very briefly, as shown in [4], the protocol guarantees atomicity because if it stops before steps 3a/3b the result is a cancel performed independently by each participant after a sufficiently long timeout. Otherwise each participant receives an idempotent request for confirmation (step 3a)/cancellation (step 3b) sent during the final phase.

3. EXAMPLE
In the following example, we show how a Web browser can implement the protocol while interacting with two airline reservation services that make use of standard HTTP.

Clients can inquire about the availability of flights at the URI: `/flight/{flight-no}/seat/{seat-no}`. For example, the `GET /flight/LX101/seat/` request will return a hyperlink to the next available seat on the flight `LX101` or none if the flight is fully booked. A `POST` request to the `/booking` URI will create a new booking resource by returning a hyperlink identifying it such as `/booking/{id}/`. The body of the request can contain a reference to the chosen flight and seat (i.e., `<flight number="LX101" seat="33F"/>`). The booking can be updated with additional information using a `PUT /booking/{id}/` request.

Seats on a flight are only reserved for a limited amount of time, during which the client should proceed with the payment to confirm the reservation. To do so, it can follow the confirmation hyperlink returned by the RESTful API (e.g., in response to a `GET /booking/{id}` the service returns `<flight number seat><payment href="/payment/X" rel="confirm" deadline="24h"></flight>`). Thus, the reservation can be confirmed with a `PUT /payment/X <VISA ...>` request issued within the given deadline, or canceled with the corresponding `DELETE /booking/{id}` request.


Book flights at low price onl
www.swiss.com/web/EN/Pages/index.aspx?Country=US
United States of America: english
SWISS
Book
Information
Book flight

Service	Resource	Deadline
swiss.com	/booking/A	24:00:00
easyjet.com	/booking/B	10:45:32

Confirm
Cancel
Continue Navigation

Figure 1: Checking the state of the current transaction with the Atomic Web Browser extension

The example shows that a RESTful service API compliant with the uniform interface constraints can make use of hypermedia to support the interaction with clients following the Try-Cancel/Confirm pattern. Clients initialize the state of a reservation with a standard `POST` request, which returns a URI identifying the resource that has been initialized. By querying the resource (with `GET`) clients can discover a hyperlink to confirm the reservation (with `PUT`) or to cancel it (with `DELETE`). Additionally, the state of the newly created resource should be discarded if a confirmation is not received by the service within a given timeout (the duration of which can be discovered by the client with a `GET` request).

Here we show a possible successful run of the protocol guaranteeing the atomicity of multiple HTTP interactions:

```
1  GET swiss.com/flight/LX101/seat -> 200
1  GET easyjet.com/flight/EZ999/seat -> 200
2 POST swiss.com/booking -> 302 (Location: /booking/A)
2 POST easyjet.com/booking -> 302 (Location: /booking/B)
2  GET swiss.com/booking/A -> 200 (Confirm URI: /payment/A)
2  GET easyjet.com/booking/B -> 200 (Confirm URI: /payment/B)
3a PUT swiss.com/payment/A -> 200
3a PUT easyjet.com/payment/B -> 200
```

The following shows a failed run of the workflow, where the protocol will perform the cancellation of the successfully completed state transitions:

```
1  GET swiss.com/flight/LX101/seat -> 200
2 POST swiss.com/booking -> 302 (Location: /booking/A)
1  GET easyjet.com/flight/EZ999/seat
-> 204 (No seat available)
3b DELETE swiss.com/booking/A -> 200
```

4. ARCHITECTURE & USER INTERFACE

The Web browser extension we have developed intercepts all HTTP requests made by the browser and looks for the presence of confirmation/cancellation links in the responses to unsafe requests (such as `POST`). To do so, the URI must be annotated with a `confirm` (or a `cancel`) link relationship and can either be found in the HTTP response header or in the body. The confirmation URI are collected transparently as the user navigates among different sites.

The extension user interface is kept very minimalistic with a single button in the browser toolbar. This button indicates the presence of an ongoing transaction (when it is activated). As shown in Fig. 1, clicking on the button opens a dialog which lists the set of resources that have been reserved and gives the user three options: 1) Continue with the navigation to perform more bookings; 2) Confirm all bookings and commit the distributed transaction; 3) Cancel the transaction. In our experience users are familiar with the notion of shopping carts where they can accumulate a set of items to be purchased within each site. The user interface we provide mimicks the same concept, even if the items accumulated in the atomic browser "shopping cart" correspond to resources across multiple sites.

If the user performs a `POST` request and the confirmation link cannot be detected the browser extension warns the user that the interaction cannot be added to the current transaction. The set of confirmation URIs are stored persistently within the browser local storage so that they can survive browser restarts. Also, a timestamp is associated with each URI to indicate to the user how much time is left before the reservations will begin to expire.

5. RELATED WORK & CONCLUSION

In addition to several threads on the *rest-discuss* mailing list, summarized by [2], the problem of transactional interactions for RESTful services has started to attract some interest also in the research community. For example, [5] proposed an approach to RESTful transactions based on isolation theorems. The RETRO [3] transaction model also complies with the REST architectural style.

In this poster we propose a browser-centric approach based on the Try-Cancel/Confirm pattern. The pattern fits well with the business requirements of many service providers that need to participate within long running transactions that do not require isolation. Thus, they offer services allowing clients to issue requests triggering state transitions (or resource reservations) which can later be canceled and have to be confirmed within a given time window.

If an agreement can be reached on the common usage of link relations to indicate the presence of confirmation/cancellation links as well as the appropriate media type representation for the confirmation payloads, we believe our lightweight approach to atomic distributed transactions for RESTful APIs has the potential to find widespread applicability due to its minimal impact on the Web architecture itself, and the great benefits it provides to users that need to complete e-commerce transactions across multiple sites atomically.

6. REFERENCES

[1] R. Fielding. *Architectural Styles and The Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[2] M. Little. *REST and transactions?*, 2009. `http://www.infoq.com/news/2009/06/rest-ts`.

[3] A. Marinos, A. R. Razavi, S. Moschoyiannis, and P. J. Krause. RETRO: A consistent and recoverable RESTful transaction model. In *ICWS 2009*, pages 181–188, 2009.

[4] G. Pardon and C. Pautasso. Towards distributed atomic transactions over RESTful services. In *REST: From Research to Practice*, pages 507–524. Springer, 2001.

[5] A. R. Razavi, A. Marinos, S. Moschoyiannis, and P. J. Krause. RESTful transactions supported by the isolation theorems. In *ICWE'09*, pages 394–409, 2009.

[6] L. Richardson and S. Ruby. *RESTful Web Services.* O'Reilly, May 2007.