An Architecture Pattern for Network Resilient Web Applications

Julien Richard-Foy
IRISA
julien@richard-foy.fr

February 13, 2014

Abstract

With the advent of Internet connected mobile devices having a an intermittent network connection rises the need for a good support of network resilience for Web applications. Achieving resilience requires to resite the application logic execution on client-side when network failures are detected and then to synchronize changes to the server when the connection is back. These failures can happen on each possible user action, making resilience a cross-cutting concern, and therefore making it hard to isolate in the application's code. This paper presents an architecture pattern based on event-sourcing isolating the network resilience concern. Applications built on top of this architecture pattern keep delivering a good quality of experience under bad network conditions. We implemented two example applications and captured the client-server synchronization logic as a reusable library.

1 Introduction

Web applications work only as long as the network connection and the server are up. If the network connection or the server is down, applications may suffer from inconsistency, freeze or latency, resulting in a bad user experience. With the advent of Internet connected mobile devices (smartphones and tablets) rises the need for a good support of an off-line mode for Web applications.

In this paper, we use the term *resilience* to refer to the ability of a Web application to deliver a good Quality of Experience (QoE) under bad network conditions. The resilience level can vary. For instance, when an application detects a network failure, it can either:

- Display an error message informing the user that its last action failed (*e.g.* Trello[1]);

[1] http://www.trello.com

- Fallback to a read-only mode (*e.g.* Google Drive[2]);
- Let the user work and try to synchronize the changes when the connection is back (*e.g.* Workflowy[3]).

To keep delivering a good QoE when the server is unreachable, the application must resite the business logic on the client-side and then synchronize the changes to the server when the connection comes back. According to the application's domain this task can be more or less hard to achieve. Typically, if the logic needs to access to a large amount of data or to perform a heavy computation, the price to pay may be too high for the client. However, in all other cases the client could take on the business logic execution. Nevertheless, the need for detecting and handling network failures on all user actions makes resilience a cross-cutting concern [1], and makes it hard to isolate in the application's code.

This article presents an architecture pattern based on event sourcing to isolate the resilience concern in Web applications. Our pattern is agnostic to the whole application architecture, it requires little adaptation for existing applications to use it, it only captures and isolates the client-server execution distribution policy and allows applications to deliver a good QoE when the network is down.

Based on this architecture pattern we implemented libraries for both the server and client sides, and used it to develop two use case applications demonstrating the isolation of the resilience concern.

2 Related Work

2.1 Offline Support

2.2 Persistent Resources

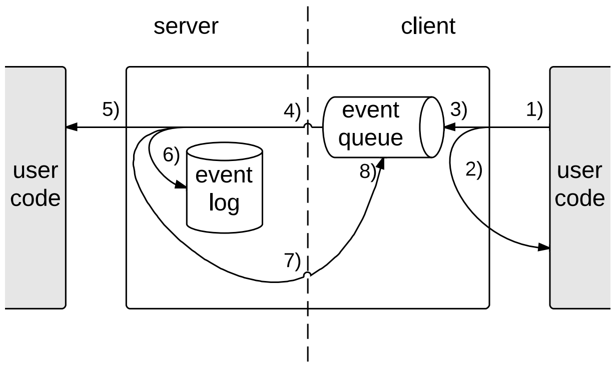
2.3 Event-sourcing

3 Contribution

Our architecture relies on the idea of event sourcing: the state of the application is determined by the application of a succession of events. For instance, in the case of a text editor, such events could be text insertion and text deletion. Events are just described by data: *e.g.* the position and the character inserted.

Figure 3 depicts the process involved when an user triggers an action in the application. The application code is partitioned between the client and server side, with a synchronization module in between. The left and right parts, in gray, represent the application code, and the central part represents the synchronization module. The process involved when a user triggers an action is the following:

[2]http://drive.google.com
[3]http://www.workflowy.com



1. A domain event corresponding to the user action is created on the client and sent to the synchronization module ;
2. The synchronization module calls back the logic defined by the application for this event ;
3. The event is put into an event queue ;
4. If the network connection is up, the queue content is sent to the server side part of the synchronization process ;
5. The synchronization module invokes the business logic defined by the application for this event ;
6. The event is stored in an event log on the server side ;
7. The application of the event is acknowledged to the client ;
8. The client removes the acknowledged event from its queue.

With this design, users have an instant feedback for their actions because their logic is executed on server-side.

4 Validation

4.1 TodoMVC application

4.2 Notes application

4.3 Limitations

5 Discussion

References

[1] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akit and Satoshi Matsuoka, editors, *ECOOP'97 Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg, 1997.