Retroactive Computing in Event-sourced Architectures

ABSTRACT

Event sourcing is an architectural style for event-based systems where state altering operations are appended to an event log. The reconstruction of prior states of the application is a well-established feature of this architecture. However, other potential benefits such as retroactive modifications to the event log or retroactive computations have not been explored in detail yet. These mechanisms enable powerful capabilities, including the computation or prediction of alternate application states, post hoc bug fixes, or the support of history-aware algorithms. After examining different approaches for retroactive computing, we apply retroaction to event sourcing and discuss conceptual considerations. We also suggest necessary architectural modifications and demonstrate how different architectures can be used for enabling retroactive capabilities of event-sourced applications.

Categories and Subject Descriptors

D.4.7 [**Software**]: Organization and Design —*Distributed systems*; D.2.11 [**Software**]: Software Engineering—*Software Architectures*

Keywords

event-driven architecture, event sourcing, retroactive programming

1. INTRODUCTION

In virtually all stateful applications, the present application state is a result of all previously executed operations. The predominant approach for organizing and persisting this application state maintains the actual values and overwrites old values by new values in case of update operations. While this approach always provides the current application state, the course of events yielding this state is not retained and thus lost. Questions regarding the history of the application state cannot be answered without any additional effort. Furthermore, queries or modifying instructions that operate on the application's history, so-called retroactive operations, cannot be aligned with this approach.

Within the last years, an alternative architectural style for event-driven architectures—event sourcing [5]—has gained some momentum, primarily due to inherent scalability and consistency properties when combined with other architectural styles. As opposed to the traditional state handling approach of updating values, event sourcing maintains an event log of all state-altering operations. Hence, the event log retains the entire history of application state and the current state can always be computed by folding the events up to the most recent event. By re-applying events from the beginning up to an arbitrary event in the event log, also a previous, historic application state can be reconstructed in retrospect.

The reconstruction of prior application states is already a well-established feature of event-sourced architectures. However, we believe that another very interesting benefit of this architectural style has not yet been considered sufficiently: By persisting the entire state history instead of only maintaining and updating latest states, event sourcing is a perfect match for a computing model where the programmers may modify and interact with the application's history retroactively in a single environment.

Thanks to distributed version control systems such as Git, the concept of retaining state-changing operations is already known to many programmers. These technologies provide access to state history externally and on a meta level. By contrast, our approach to retroactive computing includes retroactive operations on the very same environment the regular application logic resides, yielding a unified programming model for both regular and retroactive logic. This approach is very powerful, but also increases complexity and paves the way for a plethora of temporal inconsistencies.

First, events might possess causalities, meaning that one event may trigger—directly or indirectly—the occurrences of posterior events. Even worse, such causalities may happen outside of the system, making them impossible to track. When events are the result of external stimuli to the system, the degree of coupling between the system and the environment is also important: External stimuli may be dependent from the state of the system. Another important aspect are the consequences of side effects. Once an event causes an irreversible action in the system or the outside world,

In order to tame the complexity of this powerful feature, we will deliberately restrict retroactive mechanisms and fo-