
Appendix A. Aggregates and Event Sourcing: A+ES

Contributed by Rinat Abdullin

The concept of **Event Sourcing** has been used for decades but has more recently been popularized by Greg Young by applying it to DDD [Young, ES].

Event Sourcing can be used to represent the entire state of an **Aggregate (10)** as a sequence of **Events (8)** that have occurred since it was created. The Events are used to rebuild the state of the Aggregate by replaying them in the same order in which they occurred. The premise is that this approach simplifies persistence and allows capturing concepts with complex behavioral properties.

The set of Events representing the state of each Aggregate is captured in an append-only Event Stream. This Aggregate state is further mutated by successive operations that append new Events to the end of the Event Stream, as illustrated in Figure A.1. (In this appendix Events are shown as light gray rectangles to make them stand out from other concepts.)

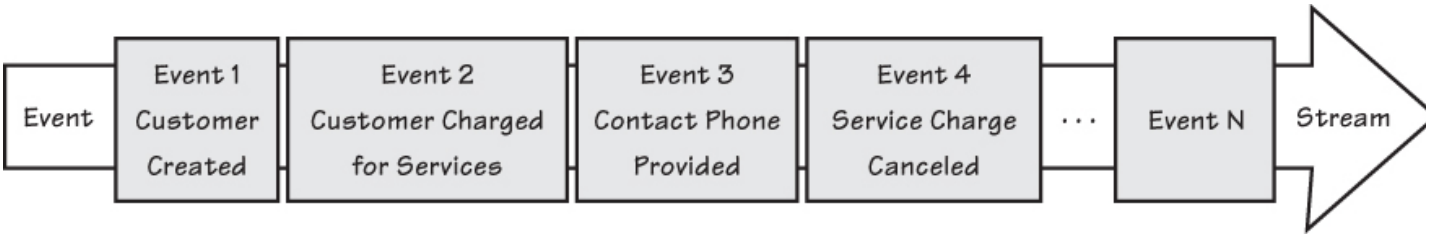

Figure A.1. An Event Stream with Domain Events in order of occurrence

The Event Stream of each Aggregate is usually persisted in **Event Stores (8)**, where they are uniquely distinguished, usually by the identity of the root **Entity (5)**. How to build an Event Store specifically for use with Event Sourcing is addressed in more detail later in the appendix.

From here forward, let's refer to this approach of using Event Sourcing to maintain the state of Aggregates and persist them as **A+ES**.

Some of the primary benefits of A+ES are:

• Event Sourcing guarantees that the reason for each change to an Aggregate instance will not be lost. When using the traditional approach of serializing the current state of an Aggregate to a database, we are always overwriting the previous serialized state, never to be recovered. However, retaining the reason for every change from the creation of an Aggregate instance through its entire lifetime can be invaluable for the business. As discussed in **Architecture (4)**, the benefits can be far-reaching: reliability, near- and far-term business intelligence, analytic discoveries, full audit log, the ability to look back in time for debugging purposes.

• The append-only nature of Event Streams performs outstandingly well and supports an array of data replication options. Using similar approaches has allowed companies such as LMAX to facilitate very low-latency equities trading systems.

• The Event-centric approach to Aggregate design can allow developers to focus more of their attention on behaviors expressed by the **Ubiquitous Language (1)** by avoiding the potential impedance mismatch of object-relational mapping and can lead to systems that are more robust and tolerant to change.

That said, make no mistake: A+ES is not a silver bullet. Consider a few realistic drawbacks:

• Defining Events for A+ES requires a deep understanding of the business domain. As in any DDD project, this level of effort is usually justifiable only for complex models from which the organization will derive competitive advantage.

• At the time of writing, there is a lack of tooling and a consistent body of knowledge in this field. This increases costs and the risks of introducing the approach to inexperienced teams.

• The number of experienced developers is limited.

• Implementing A+ES almost certainly requires some form of Command-Query Responsibility Segregation, or **CQRS (4)**, since Event Streams are hard to query. This increases developer cognitive load and learning curve.

For those undaunted by these challenges, implementing with A+ES can provide a lot of benefits. Let's examine some ways to implement using this powerful approach in the object-oriented world.