Domain-driven design: Event sourcing

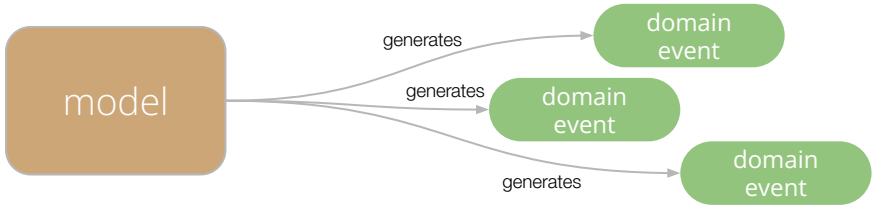
Eleonora Ciceri - 28 January 2022

Event-sourced domain model pattern

The **event-sourced domain model pattern** is based on the same premise as the domain model pattern:

business logic	complex, it belongs to a core subdomain
tactical patterns	same as domain model (value objects, events, aggregates etc.)

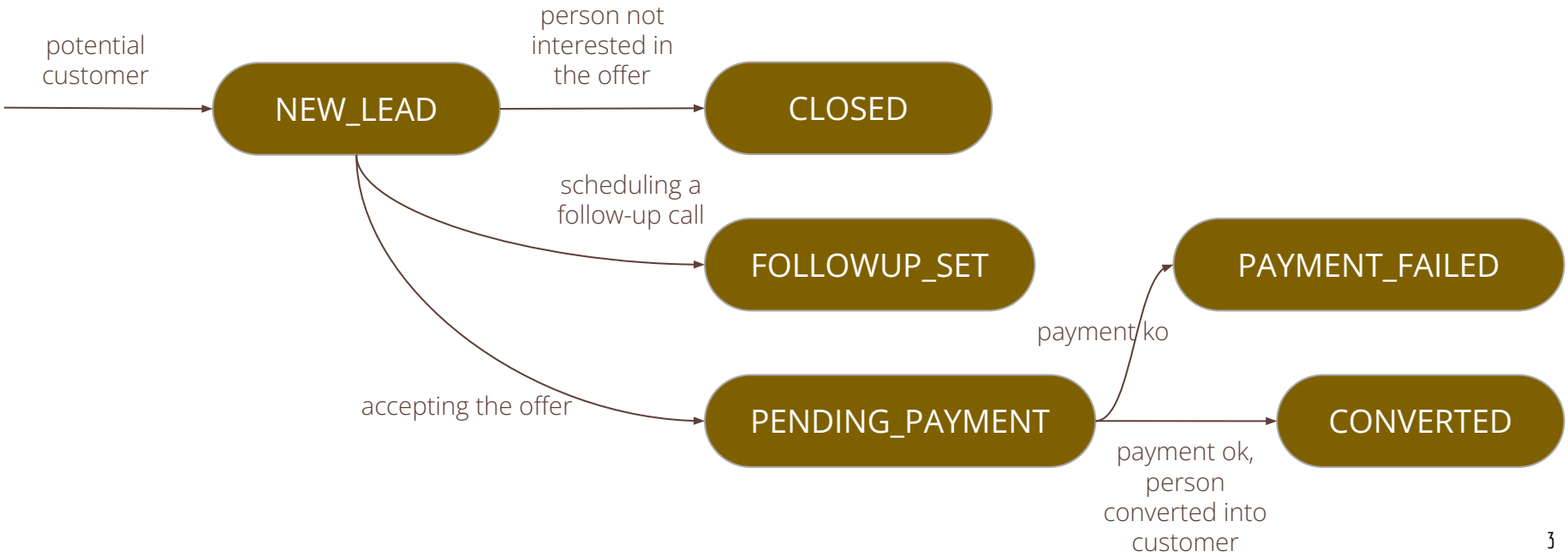
The difference lies in the way the aggregate state is persisted. In the event-sourced domain model, instead of persisting the aggregate state:



The events describe the **changes in the model**, and are thus used as **source of truth** for the aggregate's data

An example: potential customer management

In a telemarketing system, the customer goes through this processing cycle:



Transcribing the process into a table

If we were required to report this process into a table, we would obtain:

lead-id	first-name	last-name	status	phone-number	followup-on	created-on	updated-on
1	Sean	Callahan	CONVERTED	555-1246		2019-01-31T 10:02:40.32Z	2019-01-31T 10:02:40.32Z
2	Sarah	Estrada	CLOSED	555-4395		2019-03-29T 22:01:41.44Z	2019-03-29T 22:01:41.44Z
3	Stephanie	Brown	CLOSED	555-1176		2019-04-15T 23:08:45.59Z	2019-04-15T 23:08:45.59Z
4	Sami	Calhoun	CLOSED	555-1850		2019-04-25T 05:42:17.07Z	2019-04-25T 05:42:17.07Z
5	William	Smith	CONVERTED	555-3013		2019-05-14T 04:43:57.51Z	2019-05-14T 04:43:57.51Z
6	Sabri	Chan	NEW_LEAD	555-2900		2019-06-19T 15:01:49.68Z	2019-06-19T 15:01:49.68Z
7	Samantha	Espinosa	NEW_LEAD	555-8861		2019-07-17T 13:09:59.32Z	2019-07-17T 13:09:59.32Z

Problems of the table approach

We have a lot of information that we can extract from the table, and we can also assume what ubiquitous language was used when modeling the table

However, the table's data documents the **current states** of leads, and **misses the story of how each lead got to the current state**

We can't analyze the leads' lifecycles only by using the table!!

A table is not able to model time

Solution: Event sourcing (1)

Instead of the schema reflecting the current state of the aggregates, an event sourcing-based system **persists events** documenting **every change** in the lifecycle of the aggregate (introducing the **time dimension**)

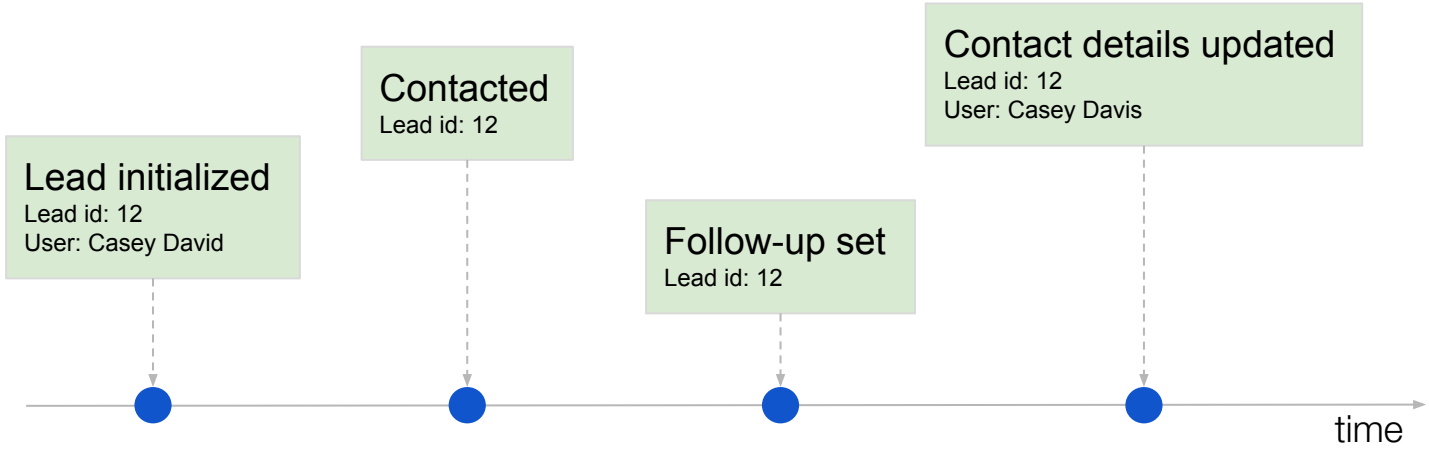

```
{
    "lead-id": 12,
    "event-id": 0,
    "event-type": "lead-initialized",
    "first-name": "Casey",
    "last-name": "David",
    "phone-number": "555-2951",

    "timestamp": "2020-05-20T09:52:55.95Z"
},
{
    "lead-id": 12,
    "event-id": 1,
    "event-type": "contacted",
    "timestamp": "2020-05-20T12:32:08.24Z"
},
```

```
{
    "lead-id": 12,
    "event-id": 2,
    "event-type": "followup-set",
    "followup-on": "2020-05-27T12:00:00.00Z",
    "timestamp": "2020-05-20T12:32:08.24Z"
},
{
    "lead-id": 12,
    "event-id": 3,
    "event-type": "contact-details-updated",
    "first-name": "Casey",
    "last-name": "Davis",
    "phone-number": "555-8101",
    "timestamp": "2020-05-20T12:32:08.24Z"
},
```

Solution: Event sourcing (2)

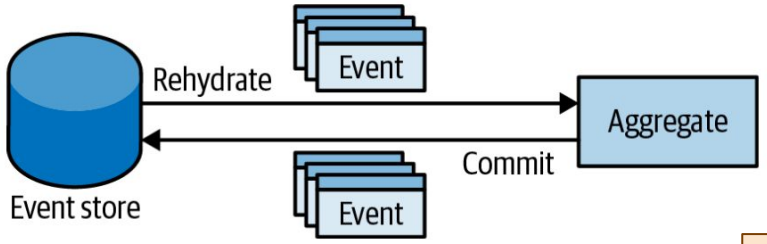
It would be like having a **timeline of changes in the aggregate**, that we can retrace to understand what happened to the aggregate before getting to the current state:



Storing events

All changes to an object's state are represented and persisted as **events**, that become the system's source of truth

The accepted name for the database that is used for persisting events is **event store**



An event store is append-only

Advantages of event sourcing

Time traveling

Just as domain events can be used to reconstitute the current state of an aggregate, they can also be used to restore **all past states** of the aggregate

Audit log

The persisted domain events represent a **strongly consistent audit log** of everything that has happened to the aggregates' states

Deep insight

You can add as many **projections of events** as you want, leveraging the existing data to provide additional insights

Advanced optimistic concurrency management

We gain deeper insight into exactly what has happened **between reading the existing events and writing new ones**. You can query events and make a business domain-driven decision as to **whether new events collide** with the attempted operation

Disadvantages of event sourcing

Learning curve

There is a sharp difference from the traditional techniques of managing data, and this demands **training of the team** and **time** to get used to the new way of thinking

Architectural complexity

Implementation of event sources introduces numerous architectural moving parts, making the **overall design more complicated**

Evolving the model

Evolving an event-sourced model is challenging, as **events are immutable**. But what if you want to change an event's schema? *(if interested, check "Versioning in an event sourced system" by Greg Young)*

References

References

[1] Vlad Khononov, *Learning Domain-Driven Design - Aligning Software Architecture and Business Strategy*, O'Reilly, 2022

[2] Scott Millett, *Patterns, Principles and Practices of Domain Driven Design,* Wrox, 2015

[3] Vernon & Evans, *Implementing Domain-Driven Design,* Addison-Wesley Professional, 2013