Event Driven Architecture

Table of Contents

[1 Event Notification 2](#_Toc80527291)

[2 Event-carried State Transfer 4](#_Toc80527292)

[3 Event Sourcing 5](#_Toc80527293)

[4 Command and Query Responsibility Segregation 5](#_Toc80527294)

Resources used:

* he Many Meanings of Event-Driven Architecture • Martin Fowler • GOTO 2017: <https://www.youtube.com/watch?v=STKCRSUsyP0>

We identified four different things; I would call them patterns that often come under the term **event-driven.** When people say they have an event-driven system they usually mean they've got at least one of these patterns in play and could mean they have all four patterns in play.

1. Event Notification

Diagram

Description automatically generatedLet's imagine some example system that might be had by an insurance company. In this insurance system we have our customers and our customers have a habit of moving around and changing their address but the insurance imagine it's something like car insurance or house insurance we're changing where your address actually affects you insurance quote levels and your premiums that you have to pay and everything so if somebody changes their address it's not just a case of updating a new dress field you actually have to trigger the insurance quoting systems come up with a new quote and they then gotta send you some mail to say “here's your new insurance agreement whatever whatever”.

The point is in a kind of simple call-and-response kind of architecture what that requires is whatever system you've got looking after customer management needs to tell your quoting system “do a re-quote”. The problem of that is that creates a dependency from the customer management system into the insurance quoting system it needs to know that it's there it needs to know what API is it has etc etc etc. That couples the knowledge of what how insurance quoting works into something that's a much more generic customer management kind of feature. That's a kind of coupling that a lot of people don't really like very much.

Diagram

Description automatically generatedSo one way we can get around this coupling is we can say instead of doing that let's try and reverse things around I don't mind my insurance quoting system knowing about my customers and that there is a customer management system there that seems a more natural dependency line to follow because something that's very very generic, managing customers, generally that's the kind of thing you want to depend on and have it depend on lots of other things.

A way we can do this is we can use an event. So, what happens is whenever some customer data changes, the **customer management system** emits an event object to some well-known place event queue or whatever and when the **insurance quoting system** can listen for that event and when the event occurs it can decide do I want to do something about it if so what do I do. That nicely we've gives us that ability to reverse the dependences. It allows the **customer management system** to be independent of all of this. This is an approach we called **event notification**. We're using events as our notification mechanism between multiple systems.

Now I've got I've described that in terms of large-scale systems independent relatively independent systems for a big company but we also use this same pattern in a very small layer system it's a classic of course approach of how to manage interactions between GUI’s and the rest of your code. When somebody enters information into a text box you don't want some very generic text box widget to have to know all about your code and which things to call instead the text box will generate an event when you change it and then your code listens to the event. It's the same idea and again the same trick is you are reversing dependencies and that's really I think the essence of why you want to do this is you want to do that reversal of dependencies. I've talked about if in terms of events and I've stressed this dependency switch but it also does something else:

* it sort of takes everything to do with the change and bottles it into a thing an object a record or something of the third that you can refer to and pass around it makes the change a kind of first-class thing.

When we call a function that's just a very transient thing we call the function supply the arguments goes away we never know that we've done it unless we've got some logging and tracing going on. When we do is something like this what we're doing is we're actually making a record of it and then we have a real thing that we can talk about refer to and pass around. That has its own benefits. We're actually getting a second benefit here which is:

This bottling it up into a thing object or I don't have to hesitate about using objects it doesn't have to be an object in OO sense I think of it as a just some kind of first-class thing.

Now that's not the only way of thinking about these those class things. Often we think of them in terms of commands as well and that leads to an interesting question

“**When do we have events and when do we have commands? What's the difference between an event and a command?**”

and I think it's actually it is both very subtle not really anything that you can say is a strict definition and yet at the same time very important. Because it's about our language and our communication and how we talk about our intentions about things. We could describe that situation I just described as the customer management system saying “re-quote insurance for this customer”. That's the CMS saying it knows what ought to be done and it's telling some remote system here's what you should do. When I phrase it in terms of **events** I'm saying, “The customers address changed”. I'm not expecting any particular response from that from anybody I'm just indicating a change both can be have this kind of bottling it up and making it as fast thingamajig effect but the way the communication patterns differ is in strongly implied as different whether you use **event** **naming** or **command** **naming** style naming. So really it boils down to this is one of the two hard problems in software right this is naming things and therefore when you're thinking about the difference between events and commands it's really how do I want to describe how the system works most accurately to the people who are reading the system.

* If I'm trying to say “I don't really care what happens as a result of this event. I don't want to care”, then yes I want to use the word **event**.
* If I on the other hand I am particular about I want something to happen then I want to name it in terms of a **command**.

Now I mean you can get all the other way. You could have your insurance quoting system insist on sending the communication and email to the to the customer and natural way to do that of course will be a command “send an email” but of course if you go too far down the event route you then get this effective passive aggressive events or passive aggressive commands where you say “the customer needs to be sent a message about their new quote! hint hint” and you know you don't really want to go down that path. If you're trying to read how the system works that's going to be confusing when you get those kinds of events occurring.

So whether you use eventual commands, under the covers they're basically the same thing. They're often messages on cues they look exactly the same the difference is all about naming that that difference is really important. Let's face it names in software systems are one of the most important things you have to deal with.

Diagram

Description automatically generated

So going back to our system again one of the **nice benefits of using events** is it allows many other systems to hook up to the event stream very easily. This is a very nice property and the result of using events for notifications here.

I need to hook up a new system that takes advantage of change needs to know about customer changing their addresses I don't have to go and talk to the customer management team and get them to add a new note of no call into their interfaces and go to endless meetings of the customer management group and two months later maybe I will get them to do something. No, they produce their events I just tap onto the event stream I'm cool. This is a very attractive ability and it'll easily allows you to do all sorts of interesting things without changing what's going on.

* Diagram

  Description automatically generated**BUT**! You get this great decoupling you get this great flexibility being able to add new consumers and you get an inability to see what's going on across the system as a whole there's no code anymore that you can just look at and it's a trade-off like so many things.

When anything sounds like it's really nice there's always a **but** isn't it? In the case of event notification that but is now you're in a situation where you've got no way you can figure out what the hell's going on in your system anymore because there's no program to look at there's no statement that says “These are all the things that happens when the customer changes their address.

The only way you can figure out what's going on if you actually have to watch the address, these various events and see what happens looking at the flow of messages through your systems try and figure out what the hell's going on. This is true in the GUI case. If you ever tried to debug a heavily event-driven GUI kind of system pretty much your only chance is to bring it up in the debugger and try and trace the various events that are flowing around.

Event notification is probably the most common thing that people say when they're event driven. They're using an event notification mechanism somewhere which is kind of odd in a way because just about every system I’ve ever looked at uses event notification to some extent. I guess when you call it event-driven it means the event notification system is seen by at least by the senior people on the project as a particularly important part of the architecture.

**TODO** – Summary, pros, cons etc.

1. Event-carried State Transfer

The second one is a minor variation on this case. So, let's go back to our little example of the insurance company again. In an event notification system all you really need to know is that the event occurred and that could really be very very generic. It could just be saying “Hey, something changed, somewhere”. You probably want to give a little bit more information on the event about that so you might say “Ah, the event thing for Linda changed” or “Something's changed about Linda's account” and then all the other systems will be triggered and say “Oh, what has happened!?”. The way they find out what's happened is they have to talk to the custom management system and even if you've got more precise about this you might say Linda's address changed though still typically the insurance quote system will say “okay, so what's the new one? What was the old one? Has it changed in a way that's important to me?”. Event notification often involves additional traffic because the the systems that are the consumer systems still have to go back to the original source to get more information and of course we reduce that burden by putting more information into the event.

If I just say “something about Linda changed” then every time you have to pile in on and it's probably after time it's not worth it does actually mean anything so it was kind of a waste of time. If I'm saying “here's the address change” then there's less traffic required because I only need to go in if I care about an address change and then I can take this to the logical final conclusion and say “can I put so much data in the events that I don't naturally need to go back to the host system” so not just do I say “Linda's address changed”, I say what her old address was what her new address is. At that point maybe I don't have to go back well maybe I do I know the address changed. I know I need to do something but now I need to go back to all of other attributes all the other things I know about Linda. Her browser history, how often she stands up as every day. All of these things we know and we can find out by asking the CMS.

But now I've got a customer management system that's got all of these queries every time something happens every other system is bombarding me with stuff. Can I reduce that, can I eliminate that load? Well I could, and this is a pattern that we tentatively named **event carried state transfer**. Which is a bit of a long-winded name but I like it because it's a play on rest right, which is resource whatsit state transfer this is event carried state this is about using events that carried dancer of state

What happens when you use this pattern is you say I want to completely forbid any

18:10

contact to the customer management system from insurance clothing system which means the insurance quoting system will keep a copy of all the data that it's ever going to need now that means of course because I'm a management system has to broadcast in its events all the data that those downstream systems are going to want to have and the downstream systems have to keep copies of everything that they are going to use I don't have to keep a copy of the entire customer record I only have to copy the stuff that I care about so it's not as bad as one might think but it's still obviously a lot of copying of data but the benefit is I now no longer need to call the customer management system at all I have all the data at hand so that might improve my performance because you know I no longer got remote network calls so the customer management system it reduces the load on the customer management system because it's not responding to all those calls it also may improve availability because if the customer management system goes down or it's intolerably slow I don't care I'll keep chugging along cuz I have all the data to myself so we can immediately see with some very nice benefits to do with that now it doesn't mean I have to do all the necessary stuff to make sure that I'm up to date I'll keep copies of the data it means of course I have to copy all the data but we're not talking about that much data these days storage use is cheap and what's more I have my insurance quoting system my availability no longer depends on the customer management system so that's a good thing now hopefully you're well enough versed in software development that you know when people say I have greater availability because I'm not dependent on something else you know what the price is what's the price of high availability lack of consistency exactly so now I've got to replicate the data I have eventual consistency to deal with that issue so I've got a consistency problem so again then carry state transfer has this trade-off I get even more decoupling even more availability but I lose complexity of keeping the duplicated data and inevitable eventual consistency so this is a less common pattern I don't know what the proportions are a lot of a time that I see event notification I don't see the state transpose state transfer but you and you do have to have event note but event notification to do this but it is a technique to bear in mind I would say this is one that you keep deeper down in your toolbox but it's many circumstances it can be a good one to play so sometimes that's what we see when we talk about event-driven systems this notification of duplicating the data to avoid calling back.

**TODO** – Summary, pros, cons etc.

1. Event Sourcing

so the third item so far what I've been doing is I've been telling you the concept and then telling you its name because I kind of think it's more exciting that way you're kind of bit more because if you read the talk beforehand you already know what the four things are so I'm really quite sure why I'm doing it but for this one I'm going to tell you what it is right away event sourcing which is a technique that is long been one that fascinated me and it's interesting to see how it's being how people are using it but let's begin by explaining what it is what do I mean by then sourcing hey by the way how many people have come across the term event sourcing stands at the sound curious okay smattering of people how many people understand it well enough they think they could explain it to somebody else whew okay I'm not going to ask you to come up on stage and do it don't worry okay these how I explain event sourcing and surprise surprise is to change your address of example again because I'm kind of boring so let's imagine I've got some kind of customer management system and I've got somebody with an address and they want to change their address in a typical system how do you change address you just say okay let's blow the old address create the new address right that's how it works right nice and simple what events or thing does is it creates this subtle change to the picture so now what we have is we still have this representation of the current state of the world but we've also got a second little area as well and when I say change my address the first thing that happens is we create an event object so this is this first class thingamajig that captures the details of the change and we've popped that first class thingamajig the event into some kind of separate storage area then we process that event and we process the event to do exactly the thing that we expected to do earlier we change the thing over the consequence of this is that we've kind of now got two sort of representations of our world we have the what I refer to as the application state which is the current state that we're thinking of the world and then we have a log of all the events that ever changed that world and this is the crucial test of event sourcing the sort of the definitional test is that at any time we can blow away our application state and confidently rebuild it from the log that's the test I mean lots of people log changes and things like that it's the ability to replay the log and confidently rebuild your state so that

…

1. Command and Query Responsibility Segregation

TODO

Some videos about how to implement such an app:

* Event Sourcing from Scratch with Apache Kafka and Spring - Kenny Bastani: <https://www.youtube.com/watch?v=pRUxU5OSB0c>