We are here with Allard Buijze at QCon London 2014. Hi, Allard. How are you?

Hi everyone. I am fine, thanks.

2.**Can you describe yourself a bit and what you have been up to lately?**

Yes. I am a software architect at Trifork and I do all kinds of projects together with a team of pretty good developers, fortunately. It is a very wide range of web applications: I have been working on electronic medical record software, I have been helping out banks on specific matters that they have, I am currently working on an emergency case management application to handle emergency calls and stuff like that. So, it is quite a wide variety of things I come across and that makes this job fun to do.

3.**You are here at QCon to speak about CQRS. Can you give our audience a quick overview of what CQRS means and event sourcing which is related?**

Yes, sure. CQRS stands for Command and Query Responsibility Segregation, which is a nice word after a few drinks, if you try to say it. Basically it is an architectural pattern that says that instead of having one monolithic API to an application, you split that into two: one for commands and one for queries - hence the name. So, it is a bit comparable to CQS which is the more object-level description - you have a method that either does something or gives you something, but not both of them. Basically it tells you to split your application interface into two components and if you want to change the application state, you use one component and if you need some information of the application state, you use the other component.

That allows you to tackle these mostly non-functional requirements, differences between the two as well, but it also allows you to choose different storage techniques for that information, for the models that you need and the information that you need. On the command side, one of the very common used techniques is event sourcing. It is not mandatory to use CQRS for event outsourcing or the other way around. They are independent techniques, but it does help you. What event sourcing does, is that instead of storing the state of objects, you store the history of all the transitions that it has gone through and that gives you a lot of power in the future to look back to what happened in the past and get new information from that which you can then use to perform better in the future, businesswise.

4.**CQRS, at least at first, it is very different from traditional web site relational database architectures. So, how do you suggest that one could try CQRS? What is the best way to start?**

I would advise anyone to just do a proof concept for yourself first. On my websites framework – axonframework.org – there is a quick start guide. It helps you to get started. It takes you step-by-step on how to start. You have to get a feeling of how it works first because most developers just want to see it work and then get a feeling of what they are doing. It is also important just to realize that it is not that different from the traditional approach. I mean it is using the same programming languages and many of the problems are solved exactly the same way. The only difference is where you send your commands and where you send your queries. Once you realize that, you have to be confident that once you can swim, you can just jump into the water and you will be fine. You will probably encounter new

environments and new problems, but most of the problems are just solved in exactly the same way as they have been in the more traditional approach.

5.**What do you think are the major advantages and disadvantages, if there are any, of CQRS?**

Unfortunately, it is not a silver bullet so there are disadvantages, but let's start with the advantages. By splitting the application into two components, it allows you to build two different models and a model is really meant to help you solve a specific problem or making decisions on what happens when you use a sense of command is a completely different problem than sending out information to a number of audiences you typically have, different kinds of groups of people wanting different types of information from your application. CQRS really helps you to build a model that is good at that and that makes a simple model and a very maintainable model as well. So, application maintenance becomes a lot easier although a bit different from the traditional approach. Another one is that by having these different models you get quite isolated components with a very clear API on them, because on the command side we have commands coming in and depending on how you want to update your query model, typically, what you see, is events coming out.

So, that is a very formal API and typically also in terms of business terms, functional terms, so everybody can understand what is going on. If you need to change anything you know exactly where to go and change it and there is one component that you need to change, typically. That's some of the advantages I encounter. Disadvantages: it is not a very widely adopted technology, so if you are stuck and need help, there are not that many places to go to. There is a mailing list for Axon users – I try to respond as fast as possible and usually I can help most of the people with my answers, but it is not a commonly used pattern yet. It is not mainstream at all. So, that is a bit of a disadvantage and it takes a different way of looking at an application. We are so used to having this one big domain model that solves everything that thinking of splitting it up into different models is something we have to get used to and that is hard.

6.**What do you think is the sweet spot for CQRS and what kind of projects could be good candidates for CQRS?**

The main sweet spot for me is how it handles complexity. So CQRS and DDD – Domain Driven Design – are quite related, although not by definition, but in practice they are very related. DDD helps you to tackle complexity but now you are splitting up a complex environment into a few less complex environments helping you even further tackle that complexity. For me, I think that is where the big power of CQRS really is.

7.**There are a lot of brownfield projects, it is not all greenfield, unfortunately. Do you think it is possible to change midcourse to a CQRS architecture and have you ever witnessed such a scenario?**

Yes. I have witnessed it once. That was on a relatively well-structured and already componentized application. I'm currently assigned on a project where there is a really big monolithic application and

trying to migrate that is very difficult. It requires a lot of trust from the people paying for the project that whatever you are trying to do. It just costs a lot of resources. You need to get a development team – the application has been built by somebody with a vision on how to get somewhere. You have to convince them to change that vision first. So you have to get the team along first. So, you would start by doing a proof of concept. Developers need code, so you give them some code to show that you can do this differently and then what you need to do is to have this migration, which is a long path of the old domain model into a split domain model. That is a very hard conversion to do and it really depends on how far you are with your project, how big the ball of mud is in that sense, whether you can do that or not.

8.**[...] If you split the command and the query, it is easy to see how you can fix things on the query side, but if you use event sourcing, how can you make it consistent both on the event sourcing and in the query side?**

João's full question: One thing that is really easy with website relational database architecture is that, well, if we have from a maintenance point of view, for instance we have a migration on the database and we introduce some bugs - it is really easy to just create some SQL and update the data to make it consistent. So, if you split the command and the query, it is easy to see how you can fix things on the query side, but if you use event sourcing, how can you make it consistent both on the event sourcing and in the query side?

That is a good question. Basically, when you do event sourcing, you should see it as an audit trail. You don't do event sourcing just for fun. It is a rather expensive type of storage because you store much more information than just the current states. So there must be a reason to do it and that is probably why a lot of banks are using it because they have this audit trail. So, if you are used to such an environment, you will know you can never change a SQL database just like that. You will always need to have some trail of what has happened.

So, in case of event sourcing, what you need to do is to take corrective actions so what you would do is to add more events, add more changes to the existing sets to counteract whatever you have done wrong with this bug. Once you do that, the query side will automatically update itself as well because that is fed from the event stream that you have from the command side. So that will go along. On the other hand, if you have a bug that is purely in the query side what you can do is – since you have the entire history in your event store, what you can do is just to throw all that away, fix whatever bug you have, throw the data away and then stream all the events back into the events into the query side and rebuild that model from the start, depending on how difficult it is to solve the problem. So, actually, instead of having less options, you have more and that might be another disadvantage of CQRS – having more options just makes it harder to chose one.

9.**As you said, you have written a CQRS framework, the Axon framework. What have you learned in writing that framework?**

Well, first of all I learned what CQRS really is. Axon is an accidental framework and it was never meant to be a framework in the initial days. I just heard about CQRS and I thought it would solve some of the problems that I was always encountering in a lot of my projects and it was just a playground for me to find out how CQRS would help. When I started doing that I noticed that much of the code that I had written had nothing to do with the, let's say, business concept I was doing the proof concept for. It was just plumbing and everything so I thought if everybody has to do that, that is a waste of time. Then I started migrating some stuff to a public repository where people could download it and before I knew it there were other people using that and I thought that was interesting. So that is when I decided to make it a framework and go on.

Since I am focusing on the plumbing, the infrastructural code sometimes told me to go back to a specification. If something needs to be serializable, you usually just say "implement serializable" and then you are done. But now it forces me to go to the specification and read exactly what it is all about and really know everything because you don't know what people are going to do with your framework. So you have to be ready for anything and building code for anything, that is a different task. That is a very interesting thing to be doing. But it has also helped me shape my views on what CQRS really is and how domain driven design can be applied in combination with that in the event driven architecture, how all that comes together and helps us build really nice and easier to maintain software.

10.**Axon allows two types of repositories: the standard one, where the aggregate state is stored and the event source that you have talked about before. So what are the use cases for each? How can I know which one to use?**

Basically, it comes to the question "Do you want to use event sourcing or not?" Event sourcing at the application level also has some advantages in the way that you can write tests. If you use event sourcing on an aggregate, on a part of a model on your command side, you have an API that consists of commands and events only, because you have the events which describe the things that happened in the past, then you have the commands expressing what you want to do right now and then you have the events coming out expressing what has been done, what has been caused by your command, what the results are. So, you can write these given when then style tests and that doesn't force you into actually using event sourcing in production. So, you can still decide to just use object relational mapping to store that model in a relational database, if you want, or in a document database or whatever kind of database you like. So that is how you chose that repository. So, if you like to store them using an event sourced fashion, you would use an event sourcing repository, if you do not want to do that then you can use a standard or regular repository which stores them in an object relational fashion so it is just really up to you to choose how you want to store your information.

11.**One thing that is talked about in the CQRS community is the difference between commands and events and at first it seems a bit weird because an event derives from a command. So why is it so important to differentiate between the two?**

The difference between the two is mostly in the intent. A command is intent of a typical user, an end user or it could be another system, to change something, to do something in your system. A command is

something that could be rejected. It could not make sense at that specific state of the application. An event is an expression of something that has happened in the past. So there is no point in refusing that because it has happened already, we cannot change what happened in the past. That notion is very important. An event from one system could be a command to another system, because an event is something that happened in system A and that could be a trigger for another application to make a decision and do something. But, by rejecting that event, you do not revert what has happened so that is a very important distinction between the two and sometimes for developers is really difficult: “Is this a command or is this an event?” All you have to do is to look at the tense. If it is past tense, it has happened and it is an event. If you are trying to do something, then it is a command.

12.**Can you explain what is “event upcasting”, which is a feature that Axon supports? Is it a common feature to use in a day-to-day basis, let’s say?**

Well, imagine if you use event sourcing because that is where event upcasting is used. You store information indefinitely or almost indefinitely because no information has ever been stored indefinitely yet. So, over time, you get new insights, hopefully. If you do not use a very formal specification or anything, the shape of events will change in the future, but with event sourcing, you have already stored an event, let’s say, from the first year of your application. It is already there so you cannot change it. That would break the auditing, your whole audit trail concept. But the way you look at the event right now is a bit different, it might contain different information, you might have insight and say: “Well, there is more information related to that event that we thought of a few years ago”. Then you can use upcasters to convert the old view on the actual events, so the real thing that happened, and change it in how you express that event nowadays. Fortunately it is not extremely common, they can sometimes be annoying to build because of where would you get the additional information from and things like that. But, in Axon, I tried to make it as easy and as simple as possible to do that and there are many ways to implement it and there are these building blocks helping you to do that.

13.**About the Axon community, what is the profile of the typical user of the Axon framework?**

I would say that the typical Axon user is a frustrated developer. They are frustrated with the complexity that they see again and again. I have met people that had the chance to do the same project twice because the first time they have miserably failed and they did not manage to complete the project because of exponentially growing complexity. So, they got frustrated with that and they said there must be a solution. That is how I got into the CQRS. They look for a solution and sometimes they find Axon and they try that in that specific scenario. They rebuild the application with half the code they have and in half the time. So that was a nice story. But, I think that the typical Axon user is a developer that is just not happy with how things are going in the traditional way and how applications just explode in terms of complexity after a few months of development.

14.**What is the future of Axon, its road map in the near and medium term?**

In the first version, in Axon 1, I focused mainly on getting all the building blocks there. In Axon 2 there were a lot of scalability building blocks, so that kept me wondering what was left for Axon 3. I noticed that a lot of the commands and events that you have - there is quite a lot of classes that you have to build, especially in Java. You can use it with Scala as well and they have the case classes which is very very nice for events because you have this single file, you have a one line description of what your command or event should look like. But Java does not have that so I am thinking of some IDE support to make it easier to just describe commands and events and it will generate the Java code for you, making it a lot easier. That is one of the things. Building more event store implementation. Currently there is a JPA implementation, there is a MongoDB implementation and I am thinking of a – I am working on a pure JDBC one and I am also working on a file storage implementation that is a lot faster than my SQL typically. I did a littlebench mark on my laptop and I can store around 180,000 events per second on my spinning disc and on my laptop which is more than most applications will ever need anyway. So those are some of the areas I am looking into: mainly usability and then improving performance and building more implementations of the building blocks that I have in Axon.

15.**So Axon is CQRS framework for Java. Are you aware of similar frameworks for other platforms and languages?**

I have heard that there is a number of implementations in the .net area. I am not too familiar with them. One is called NCQRS and I spoke to the developer of that and I know his vision on CQRS is quite similar to mine. I do not know how the framework is, so I could not compare it to Axon in that sense. Then there is – I think it is called Lokad and I do not know what they focus on either, but in the .net area there is a choice of around five frameworks and I know in the Java scene I only know of Axon and there is EventSourced which focused on events sourcing only. I do not know how that is maintained. Those are some of the technologies that I know about.

16.**Any final thoughts for our audience?**

I would advise anybody to just give it a try and to find out for yourself whether it is a suitable technology and not to get distracted by the seeming complexity that it brings. It is a lot simpler than you think if you just stick to a very simple rule about where commands go and where queries go. Just give it a try and see if it works for you and if it doesn't, no hard feelings.

João: Thank you very much, Allard.

Thank you.