How I ended up having to guarantee that a business state could cross the Atlantic in 1 millisecond

Facing extremely complex technical challenges, our reflexes as developers can sometimes play tricks on us, making us miss out simple solutions. DDD guides us towards identifying simple solutions by seeking inspiration more often than usual from the business people.

Foreign exchange market (FX) for the big ones

I remember a case twelve years ago, when a simple discussion with the business had made it possible to solve a technical problem that initially seemed unsolvable (related to the speed of light). At that time, I worked for a large investment bank.

When you are a large company or an investor and you want to convert a large amount of one currency (ex: 3 Million EUR - €) into another (ex: USD - $), you generally go through a stock exchange specializing in this type of operation on the foreign exchange (FX) market. When you are a big investor or you often have this kind of need, you can even set up a dedicated connection with your preferred supplier (like the bank I worked for) to get better prices, faster.

My end-users at the time were Market Makers on the foreign exchange market. In other words, they were specialized traders who offered to sell and buy assets (here currencies) with publicly quoted prices. Against that risky activity which is not necessarily profitable but necessary for the financial markets, the banks which are doing market making obtain other benefits elsewhere (later, during public IPOs for instance). So, it was an important activity for my employer. One of the challenges of this activity is to respond to all currency exchange requests as quickly as possible without losing too much money to predators (some investors, sometimes other banks).

To do their job, these traders needed different tools:

- **Pricers:** which constantly readjust and re-compute prices based on input market indicators serving as benchmarks
- **Streaming / contribution systems:** pushing on the markets these prices are calculated by our Pricers every time they changed (several times per second). They also add relevant margins depending on the targeted customers categories; that we call *Tiering*
- **On-demand price streaming and automatic negotiation systems:** for specific amounts to customers who request it (this is called Request for Stream - RFS)
- **Systems to manage executions:** i.e. whether or not to accept a deal request from a client based on a price sent earlier
- **Hedging systems:** to invest in position intended to offset potential losses or gains that may be incurred by their activity
- **And a whole arsenal of control systems** necessary during executions to avoid fraud, abuse and excesses from certain customers or competitors

Given the requirements of the foreign exchange market in terms of latency (from a few hundred microseconds to a few milliseconds only), most of these functions were fulfilled by the software that we had developed for them. As a result, the market makers could easily configure their systems and let them work all day long in auto and low latency mode. The declutching of the automatic mode towards a manual intervention of a trader on the desk are very rare and happen only for critical cases (huge quantities, exotic currency involved or any other risky situation).

Low latency and Reactive SOA

Back in the days I was the chief architect to design the new Market Making platform of the Bank for the FX Spot assets (i.e. currency pairs). We were developing all these needed systems (Pricers, execution engines, connectivity gateways, hedge engines, credit check service, deal capture backends, etc.) with various Dev teams. I had fostered a Service Oriented Approach (SOA) at the time, but with low latency (i.e.: sub-millisecond) protocols, middlewares, infrastructure, network and implementations. We were already fully embracing reactive programming and event-driven models that were common in finance (or Telco) but which is becoming more and more fashionable these days, especially in the Domain Driven Design (DDD) community.

Concretely speaking, every time a key-indicator was changing on the markets, our corresponding Pricers triggered re-computations. Every new computed price was then pushed towards subscribed customers who were in a "cancel and replace" consumption mode (at the middleware level). It is as if our traders are constantly sending new mini quotes (prices) to subscribed customers. At any time, a customer could decide to make a request to us to execute one of the prices he had received.

Some markets allow you to send firm orders or prices (i.e. which obligatorily commit those who send them). Our Traders on the FX market had the ability to inspect, control and possibly refuse every execution request sent by a client. Indeed, out of the hundreds or thousands of price updates that we send to a customer, we don't always want to make a deal on our old prices, especially if the rug was pulled out from under the market in between.

Execution: a critical step

Execution is a critical phase where you have to go fast (ideally under a millisecond, but not more than 3 milliseconds (ms) in total, network latency included). If we are too slow to carry out this execution, we then take the risk of spending our time refusing executions to our customers because market prices have moved too much between the time they/them received our price, and the time when we answer him/her.

To have an idea of the effect it can have on the customer, imagine yourself alone in a desert area, midsummer. It is extremely hot. You are terribly thirsty and a distributor of cold drinks which is there by chance offers you a large bottle of water for 5 cents. 5 cents only for 2 litres of fresh water! You are happy like everything, you put your coin... but the machine refuses to sell you the bottle and returns your coin. You try again: same thing ... you try again: another refusal of the machine. And again, and again... Well... that's how we lose customers forever ;-)

As I said, we only have a few milliseconds to orchestrate in parallel a set of controls, protection mechanisms and verifications before we can say yes to the client.

Among these:

- **Last Look:** to check that our price is always in line with the rest of the market by comparing it (to avoid being eaten by predators who exploit us being slow to change prices)
- **KYC & black listing:** to check if we are not dealing with a bad client or someone we don't want or can't do business with (embargo, terrorism)
- **Rate limiting:** to check that this client is not knocking us out with too many consecutive execution requests. Otherwise, it is refused certain executions if they are too close to each other.
- **Credit Check**

Credit Check is the new black

Finally, we also have to check the current creditworthiness of the customer. To do so, we check that this execution - if we accept it- will not consume more than the maximum amount we authorize for him or her within our bank. To take an extreme metaphor, one can see a customer's line of credit within our credit check much like a kind of maximum amount for an authorized overdraft.

The service that our execution orchestrator must request to do this type of verification is called Credit Check and was one of our internal services.

By engaging in credit checking before real transactions take place, confidence is maintained that each party has enough credit to carry out and honour the deal. The Credit Check service is responsible for answering the question: given everything that has already been executed by this client with our services (i.e. his current remaining line of credit), do we still have the right to do a deal of x amount of USD with him or has he already exhausted all of his credit lines with us?

Every time we say yes and a client makes a deal with us, we also decrease the amount of his remaining line of credit. If not, we return a deny message to the customer, indicating that he has exceeded his credit limits with our establishment.

To sum up: in addition to answering all these questions in less than 3ms, we have the state of the remaining credit lines for every customer to maintain in a reliable, fast and transactional manner.

Can't we speed up that check?!?

Actually, it starts to get more complicated when we realize that **the same client** (a large international group for example) **can execute us simultaneously on different continents ...**

Indeed, on the foreign exchange market, our platforms are open 24/7 - 6.5 / 7d and have connectivity points all over the world (New York, London, Singapore, Hong-Kong, Tokyo). In concrete terms, this allows our customers, who also have systems in all these places, to make execution requests to us simultaneously on the same currency pair (EUR / USD for example) in

NY, London and HK. To simplify the following explanations, we will pretend that we only have 2 connectivity zones to manage here: NY and London.

As a reminder, **a round-trip latency between London and NY** when everything is going really well for you **is below 60ms**. So that's almost 60 times more than the maximum latency allowed for one execution request... being one of the various steps of the execution itself... Houston, I think we have a problem.

Problems are now becoming: *how to maintain a mutable and globally shared state of credit limits for this client worldwide, while keeping acceptable response times for his executions (i.e.: less than 3 milliseconds)? Where should the Credit Check component that contains this worldwide state for every customer's credit limits be deployed?*

Well in fact the problem is not: *how to maintain a shared mutable state worldwide for this client by defying the laws of physics and the speed of light?* Doing so, we probably would have asked ourselves silly questions such as: *"How about locating the Credit Check Service in Iceland (between London and NY)?"* or *"How about we update the worldwide Credit Check status of our customers by trying to use quantum entanglement?"* ;-)

This is indeed what happens when tired or when trapped in a mental model that is a little too simple (trying to optimize every part thoroughly instead of reassessing the problem from the beginning).

No, the problem here is rather to *find a way to avoid overshooting credit lines for a client, despite the executions that they can launch in parallel on different sites* (London and NY for example), *while continuing to ensure that our systems on each site are not slowed down and respond well to each request for execution in less than 3ms.*

When it is complicated like this, it is usually time to discuss with the domain experts in order to find a hint or a trade-off that is sufficient and acceptable to solve our problem.

Err ... we need to talk

Here, the real problem we wanted to avoid was rather understood by talking more with our business. Like always.

During one of these discussions, we've discovered that our Credit Check service should be here especially to prevent us from situations like: a client on the verge of bankruptcy arranges to send 2 large execution requests at the same time on 2 continents in order to leave us alone with all his debt before we even had time to realize that he tricked us (by making us execute 2 times his maximum authorized amount).

A race condition that can cost us dearly if it occurs ... Formulated like this, things become much more doable. So how can we avoid such a situation? I will give you a few minutes to search for a solution on your own before you continue reading to discover ours.

(...)

Ok. Did you take some time to think about it a little bit? Now I will tell you what we found at the time, to meet our goals.

A nice-enough trade-off

Since we couldn't yet challenge the speed of light (or even exploit quantum entanglement unfortunately ;-) we mitigated the risks otherwise. We had split the total amount that a client has the right to execute with us by the number of places where he or she can execute with us simultaneously (i.e. where we deployed our Market Making platform with local Credit Check service to maintain acceptable latency). Doing so, if the customer executes us in parallel on each continent, he will only be able to spend his various local subtotal lots of his global amount.

Let's take a concrete example here. Let's also consider only 2 geographical sites for the sake of simplicity: NY and London.

If a client has a remaining of 100 Million credit limits with us, we divide his remaining credit line amount in 2:

- 50 Million max allowed for execution in NY
- and 50 Million max allowed for execution in London.

The split of every client's global credit limits over several local geographical subdivisions combined with our local speed limiting protections (avoiding any client to do multiple local executions consecutively) was giving us enough time to consolidate the client's worldwide line of credit before another executions from him could happen in the same location (with an average latency of 200 milliseconds between two local executions). This saved us.

That sounds obvious putting like this. But it was really difficult for us at the time to go beyond our embarrassment of not having found a tech-oriented solution before we talked to the business stakeholders. **We had got ourselves fixed on a naive question**: "How to reduce the intercontinental latency to keep the credit check sync" under 1 millisecond?).

Of course, this decision could only be taken because the business impacts of it were assessed and accepted by our business stakeholders. This is an explicit compromise for everyone on the side of the bank. Not a decision taken unilaterally by the IT.

As stated at the beginning of our story, a badly stated problem can have serious consequences on the implementation side. How to hope in this case to offer a solution that is suitable and not a technical gas plant just-in-case the need might be a little bit here, a little bit of that ...

Conclusion

Here, we could only find an acceptable trade-off to this complicated problem by talking more to the domain experts (once again powered by the 5 Why technique). It saved us from serious operational losses or very expensive infrastructure (Credit Check deployed in Iceland or quantum entanglement? C'mon!)

Of course, in our case the search for a purely technical solution did not last very long this time because we all knew that we were constrained by some strong constants of the universe.

But I can bet my round here that **almost every DEV person has already experienced situations where we were trying to find solutions to complicated problems, we thought we have...** before taking the time to dare asking questions to the business domain experts. This story is an example among others of how closer interactions between the business and Dev, lead to trade-offs that satisfy everyone. **The kind of interactions that Domain Driven Design (DDD) encourages you to foster as often as possible**. In other words here, if we had stayed on our pure technical field without trying to understand the business stakes, we would have never been proactive enough and a real partner to solve this problem for the business.

Obsessed by use cases and problem space challenges since a very long time (Vs. solution-driven approaches), **Domain Driven Design** was a confirmation for me. **An invaluable ally to continue trying to align my solutions with the real problems of the people I work for**.

This involves many discussions and closer relationships with the domain experts. And it's also a mercy-less struggle to fight our technical biases, initial mental models and the ubiquity of implicitness in all our interactions with others.

Thomas PIERRAIN

International speaker & trainer, eXtreme Programmer and architect obsessed by use cases and the problem's space, Thomas animates for more than 3 years the meetup **#DDDFR** and share with his customers his experience on Domain Driven Design and Software Architecture as well as his other specialties (event sourcing, bi- temporality, reactive programming, multithreading, low latency middleware, Outside-in TDD, XP, Event Storming). Enthusiast promoter of Hexagonal Architecture for years, Thomas has also written and done lots of live-coding on the subject, especially once on stage with Alistair Cockburn.

After working 12 years in market finance (over 20 years of experience), Thomas created the **42 skillz** company with his friend & associate Bruno Boucard and continued to discover various domains such as Online gaming, Transport, Finance, Hospital care. For the past few months, Thomas has enjoyed discovering the Hospitality industry with one of his recent clients.

References:

- Reactive Manifesto: https://www.reactivemanifesto.org
- Pragmatic architecture: http://tpierrain.blogspot.com/2013/04/the-pragmatic-architect.html

Doc version 1.5

Special thanks to the 👍 reviewers:

Nick TUNE, Arnaud LEMAIRE, Jean-Baptiste DUSSEAUT, Wassel ALAZHAR, Yi HAN, Yannick GRENZINGER & Cyrille DUPUYDAUBY