DAML: The Contract Language of Distributed Ledgers

case study

A DISCUSSION BETWEEN SHAUL KFIR AND CAMILLE FOURNIER

When Shaul Kfir cofounded Digital Asset in 2014, he was out to prove something to the financial services industry. He saw it as being not only hamstrung by an inefficient system for transaction reconciliation, but also in danger of missing out on what blockchain technology could do to address its shortcomings.

Since then, Digital Asset has gone to market with its own distributed-ledger technology, DAML (Digital Asset Modeling Language). And that does indeed take advantage of blockchain—only not in quite the way Kfir had initially intended. He and Digital Asset ended up taking an engineering “journey” to get to where they are today.

Kfir readily admits his own background in cryptography and cryptocurrency—both as a researcher (at Technion and MIT) and as a cryptocurrency entrepreneur in Israel—had more than just a little to do with the course that was originally charted. As for lessons learned along the way, Camille Fournier, the head of platform development for a leading New York City hedge fund, helps to elicit those here. She brings to the exercise her own background in distributed-systems consensus (as one of the original committers to the Apache Zookeeper Project) and financial services (as a former VP of technology at Goldman Sachs).

CAMILLE FOURNIER One of the proposed applications of blockchain for the finance world now focuses on how it might be used to solve the distributed-ledger problem. How would you say that challenge is generally viewed at this point?

SHAUL KFIR Let's start by considering the current state of IT in the financial industry. As any new product is introduced to this highly interconnected ecosystem, each organization builds a system to handle the new product's workflows. For each financial institution, this will become only one of many similar systems—so similar, in fact, that they duplicate many of the same functions. Each of these systems, in turn, is integrated into other internal systems that all touch the same accounts, the same assets, and the same reference data. Beyond that, each organization is also faced with building reconciliation processes with each of its counterparties.

This already presents two major problems. First, if there are n new systems in the industry—one for each organization—you're soon going to end up with something on the order of n^2 new bilateral reconciliation relationships between the various entities. Second, you end up with a new system built for each new product instead of mutualizing the existing infrastructure and making it possible for products to be built and composed on top of each other.

As the financial industry or any product in it continues to mature, the natural tendency is to centralize in a hub-and-spoke manner around established central infrastructure providers so as to revert to something more like n different relationships, where everyone can just

focus on reconciling against the infrastructure providers at the center of the network.

Another thing to bear in mind here is that the way these systems are built typically provides only for reconciliation to happen in batch mode as an aggregate of all the transactions that have taken place over the course of the day. This is a historic remnant of how the financial system was originally designed and falls well short of the demands of today's financial world if only because there's too much intra-day risk associated with waiting until the end of the day to know with certainty what assets and liabilities you have.

CF You're saying the current model relies too much on centralized reconciliation providers and increases intra-day risk by waiting until the end of the day to do reconciliation. How is a distributed ledger for transaction reconciliation going to address both of those concerns?

SK Actually, there's a third concern I need to mention—and this is the most critical one to solve. As this ecosystem of distributed workflows continues to grow in complexity, it becomes more and more difficult to drive innovation, since each change forces each entity to make adjustments to its own internal systems and processes. The pace of change, as a consequence, eventually grinds to a halt.

Try to imagine a distributed ledger as a virtual SQL database that the whole industry has access to—filtered, of course, such that each company can see only what it's allowed to see. It's with this mental model that distributed ledgers manage to address these problems in two main ways. One is that they reconcile the different companies' ledgers on a per-transaction basis in realtime such that

one shared database can show each of those companies exactly where it stands at any point. This means you don't have to wait until the end of the day to confirm your position.

The second major advantage is that a distributed ledger transforms the rollout of new workflows into a lightweight process. Imagine a company suggesting a new workflow simply by adding new tables and stored procedures to this database—which is to say, by doing something that resembles what SaaS [software as a service] companies currently do to provide for a continuous rollout of features. This will reduce friction and, as a consequence, lead to a faster pace of innovation as existing workflows become composable elements of new workflows.

To clarify, think about how large technology companies use their infrastructure to achieve greater agility. Most of them today have some logically centralized infrastructure that includes a central code repository and a CI/CD [continuous integration/continuous delivery] system. If these ideas can be expanded to an industry level in the sense that you can start rolling out workflows as smart contracts that are written only once and then made available for everyone to build upon, that's clearly more efficient than leaving it to each organization to write its own workflows.

CF Fine, but now let's back up for a second to talk about what actually distinguishes a distributed ledger from the classic reconciliation approach these large institutions at the center of the financial network are currently following.

SK As things currently stand, these large institutions hold the ledgers of record for many different markets.

There are at least two problems with that. One is that the systems used by many of these infrastructure providers are old—which is to say you can't use an API to go in and get a realtime view but instead are forced to wait for that night's batch to be run. Distributed ledgers don't address this problem directly, but they've been designed explicitly to take advantage of modern technology.

Then there's a second issue here that's more fundamental. A large financial institution can't be in the position of needing to trust an API call into someone else's infrastructure to learn that something has just shifted on one of its trades by a few million dollars. Each financial institution holds its own ledger of record that it can make reference to whenever needed. But then, with at least one entity on either side of a trade, this means there will be at least two ledgers in play, both of which are going to require reconciliation around each event since they both touch the same data. A distributed ledger synchronizes these separate ledgers as if they were part of the same database, while, in fact, each is controlled by one of the institutions that's party to the trade.

CF So, this isn't just a problem with outdated technology. At least theoretically, you're saying a centralized reconciliation provider ought to be able to take advantage of newer hardware to become faster, more API driven, and less batch driven. But it seems the counterparties don't really trust the centralized third party to hold all this information on their behalf. Instead, what they really want is to have their own copy of that data, no matter what.

SK Right... because they need to verify, as well as trust.

What
block-
chain
brings
to the
equation is
complete trust

CF How does a distributed ledger differ from a centralized reconciliation system?

SK First off, a distributed ledger is, of course, distributed. Each institution has a local ledger that's specific to that institution's assets. In this sense, the ledger is distributed so that each entity is able to view those shards of data that are relevant to it. At the same time, the integrity of the system provides the assurance that—if entity A, entity B, and entity C all have some specific data—their views of that data and the positions they hold with respect to it are consistent.

Another distinction is that most of the distributed ledgers today incorporate blockchains. But, over time, I think we'll see more non-blockchain distributed ledgers that still manage to address these goals of consistency and infrastructure mutualization. What blockchain brings to the equation is complete trust—that is, how does each entity know that its view of the ledger represents the truth, the whole truth, and nothing but the truth?

Getting the truth from an engineering perspective is easy enough. A hash is a commitment to the veracity of some data.

As for “nothing but the truth,” there are signatures to authenticate that the data is correct.

But then there's “the whole truth.” This is where the blockchain structure comes in, with the append-only hash chain providing one means for ensuring that.

It isn't the *only* way to accomplish this, however. We need to distinguish between the technical meaning of blockchain—which refers to an immutable data structure that maintains an append-only chain of changes—and the

more popular industry usage of blockchain, which can refer to anything having to do with distributed ledgers or, for that matter, anything that might be used to track the history of some distributed service. In fact, that's why we've come to use the term *distributed ledger technology*—or DLT—to imply this broader view.

CF You've been careful to point out that the blockchain data structure is only one of the options available. Is there some reason to consider other options?

SK Actually, there is a problem with blockchain. Say we have two different companies that decide they want to do a swap, where one company lends one asset and gets some other asset in return. Each asset happens to be managed by a different ledger. One way to deal with this would be to use two-phase commits in much the same way cryptocurrencies employ hash locks. But this really puts the onus on the application developer or on whatever middleware you might happen to have in place. An alternative would be to ensure that the distributed ledgers are composable and able, by design, to merge and split seamlessly. There are a number of approaches for accomplishing this.

Conceptually, a blockchain design is just a single chain of hashes and so doesn't really lend itself nicely to network composability. If you want to use a blockchain without sacrificing network composability, you need to take a completely different approach. I think it's critical for a distributed-ledger network to have this sort of composability.

CF Before diving too deeply into the blockchain side of this,

I'd like to know which parts of the centralized reconciliation problem proved to be especially challenging for you to address using this distributed-ledger approach.

SK First, I should provide a bit of context. For one thing, it's important to acknowledge that consistency, data privacy, and liveness are all issues that become quite nuanced in any distributed system. Then there are a few things that are more specific to distributed ledgers. We have the challenge of ensuring confidentiality in the sense that people should be allowed to see only that slice of data that pertains directly to them, while also being assured that the integrity of the overall data model will be maintained. That's a nuanced computer science problem that involves the use of formal analysis of the protocols.

The second thing—and I don't think this problem is widely acknowledged in the industry—has to do with what I call the independence of SLAs [service-level agreements], or *liveness* in computer science terms. This means that, in a multiparty workflow, you don't want system downtime for a single party to hold up the overall process. For example, once an exchange matches a buyer and a seller—known as an *execution*—that's legally binding. If we were trying to employ the model here that most blockchains use, the exchange would report the trade by coordinating a transaction with a multiparty signature workflow. A minimum of three different parties would have to sign to acknowledge the trade: the exchange, the buyer, and the seller. And, in practice, there are always more than three parties. In fact, you have at least seven different parties that need to sign each trade execution transaction among all the different clearing participants, custodians, and so on.

Still, from a legal standpoint, if one of these parties' systems happens to be down or unresponsive, you don't want to hold back the whole workflow, since the trade is legally binding and needs to be entered into the ledger. And maintaining liveness—while also making sure that, if something needs to go through, it goes through—is a different problem to solve. To deal with that, we essentially had to build a very fine-grained delegation model similar to the OCAP [object capability] model.

Among the more significant lessons Kfir and his team at Digital Asset learned as they were striving to build a new system for transaction reconciliation was that they actually were dealing with more than just a distributed system problem. Much more, it turns out.

To their surprise, they found that each of the financial institutions they encountered had gone about implementing the industry's specifications for transaction reconciliation in a somewhat different fashion. This resulted in values in end-of-the-day reconciliations that didn't necessarily match from one institution to the next.

That's how they learned they also had a major data-modeling problem on their hands.

CF Tell me more about the role blockchain plays in your distributed ledger.

SK We don't use a blockchain data structure per se. Blockchains at this point work mostly with tokens. Whether it's Bitcoin or Ethereum or whatever, they employ tokens

The problem is that tokens are the digital equivalent of a bearer instrument in the financial world.

that are inherent first-class citizens. But that model quickly starts to break down as you add even a small amount of complexity.

The problem is that tokens are the digital equivalent of a bearer instrument in the financial world. They're simply unable to capture granular rights. For example, with a stock, the owner has the right to transfer, while the share registry has the right to split or merge. Cryptocurrencies and similar tokens don't capture these sorts of rights.

And then things become even more complex with corporate actions like voting rights. This is how we learned we would need to come up with an abstraction layer that isn't token based. Which proved to be quite challenging. With that said, one of the Bitcoin elements we did end up keeping is its state-management model where each transaction consumes contracts and then creates new ones.

People often ask me, "Why did you feel the need to build a new contract language or a new abstraction layer?" I tell them we basically were forced to look for a new paradigm.

Just as the REST [representational state transfer] API didn't exist prior to 2000 for the simple reason that nobody really needed a RESTful architecture until the web came along, we found ourselves in a similar situation. We were tackling a new sort of problem—a workflow problem—and found we needed to come up with a new language and new abstraction layer. And we learned not to be afraid of doing that.

Interestingly, driven by customer demand, we've ported our language over to traditional SQL databases, because people believe that offers a powerful abstraction for

modeling assets and workflows even in situations that don't really require the distributed aspect.

CF OK, but I'm still wondering what initially made you think about blockchain as a potential solution to your reconciliation problem. What suggested blockchain would be an obvious fit?

SK It was just my personal naivete. We have other people in the company who came from other disciplines, but my background is in cryptography and cryptocurrency, so I was still drinking the blockchain Kool-Aid in 2012. This led me to think, "These bank IT people probably just don't know what they're doing, so I'll show them how blockchain can make things better."

CF That sounds somewhat familiar. I'd say you weren't alone in drinking that Kool-Aid.

SK As we started to dig in, we spoke with people in many different roles at a number of financial institutions and started to see some troubling patterns emerge. One was that the pace of innovation throughout the industry was sluggish at best. That's what first led us to think, "OK, maybe blockchain actually *does* make sense here, given that a smart contract language could really help these people roll out something faster."

But that's also when we really shifted our focus to a more holistic view: "How can we make this industry more productive and agile?" Which completely shifted our value proposition. Our mission today is to accelerate the pace of innovation in these industries by building a powerful abstraction, along with a powerful set of developer tools.

Another factor that really surprised us at first had to do with the problem of garbage-in, garbage-out data.

Basically, as we were trying to figure out what we should do, we found that the data structuring and message structuring across virtually every asset proved to be just horrible. Everyone seemed to have a different view of how the data ought to look and, while most of the industry standards spelled out specifications for messages between participants, they didn't address the semantics. So, everyone had implemented the specifications for transaction reconciliation differently. There were all these variances from the low level on up. This led to end-of-day reconciliations where the values didn't actually match from one institution to the next.

Once we saw that, we knew this wasn't just a distributed-systems problem, but also an issue that involved the abstractions and data structures. That's when we started to approach things in an entirely different manner. As a company, we realized we were tackling a problem that was bigger than we at the time were equipped to handle.

Once we accepted that, we joined forces with another company, Zurich-based Elevance, which we later ended up acquiring. Elevance was made up primarily of programming-language experts, formal-methods experts, and ex-quants who had been working on the very same data-modeling problem we had been puzzling over, and they had come to realize their approach was one that would work particularly well on a distributed ledger. We, on the other hand, felt pretty good about our distributed-ledger work, but we knew we needed help with the data modeling. As it turns out, we were both quite fortunate to find each other.

CF So, you put your distributed ledger together with the data-modeling work from Elevance to more or less solve your system-design problems. But what did that leave you with from all that blockchain work you'd initially done on your distributed ledger? What portion of that work has proven to remain stable and at least somewhat useful to you?

SK The most important work that remains from our early days is a robust abstract model of a distributed ledger. That's to say, in the course of designing the interface between an ideal domain language and a practical distributed ledger, we had to design a formally specified abstract model of a blockchain that included a data-structuring model, an integrity model, and a privacy model.

While we initially built all this for our own internal needs, it proved robust enough to let us later integrate DAML with numerous other distributed ledgers, including not only our own but also others from VMware, Hyperledger, and more. This also allowed us to have a common language to use in presenting to users the tradeoffs between different DAML platforms.

CF Did you encounter any other problems that you think are going to cause some real grief in the blockchain world?

SK There are situations where it's going to prove difficult to get the blockchain tokens to match up. Tokens simply aren't expressive enough to represent any application that presents even moderate complexity. That tends to get lost in all the blockchain hyperbole.

There's also a much bigger problem we needed to deal with—though this doesn't appear to have been much of an issue on many other platforms to date—and that is

sub-transaction privacy. Even within a single transaction, different entities have different rights as to what they can see. For example, if you and I were to swap a share for cash, our banks should see only a cash transfer, while the share registry should see only the share transfer—even while the two of us are able to see the whole transaction. If there's some other blockchain that's able to accomplish this, we're not aware of it.

Another big revelation for Kfir and his colleagues came with the realization that the problem they were addressing was less a technical one than the general burden of ever-growing complexity. Which is to say that, rather than building custom solutions the institutions themselves could then deploy, the focus should instead be on creating tools that programmers at each of the institutions could use to create their own solutions.

With this, the emphasis shifted to creating a synchronization layer to provide for consistency among all the trading partners, and complementing that with a contract language and an abstraction layer with enough support for system functionality to let developers at each institution focus more on business logic than on more traditional programming concerns.

CF You mentioned your decision to partner with—and ultimately acquire—Elevance in order to take advantage of its expertise with formal methods and programming languages. What drove you to make that decision, and do you think the absence of just such expertise may have had

something to do with the failures that have occurred over the years in the distributed-ledger space?

SK I wouldn't be so quick to label those previous efforts as failures. I just think of them as things that happened back in the early days of distributed ledgers. Remember that the first of these projects started up in the 1980s, well before the Internet boom. In the same way, I don't think we failed initially, either. We moved forward in one direction before recognizing that our initial model, which bore a lot of similarities to cryptocurrencies, was one that many people would have difficulty working with as complexity grew over time. Basically, you would end up needing a Ph.D. in cryptography or security to write your distributed-ledger applications. Which obviously wouldn't get us very close to achieving our goal of increasing productivity and accelerating innovation.

We started out with a model that was very token-based but then realized we needed to be more workflow-centric as we faced growing complexity. In effect, we moved to a model in which you have different action types for the different sorts of things you need to do, rather like Ripple transaction types. [Ripple is a financial-transaction protocol based on a shared public ledger.] This has worked well in Ripple's case since that's focused on a very specific domain—namely, payments. It could put the emphasis on defining certain types of transactions as the sorts of things you're allowed to do with your system.

As opposed to Ripple, though, we weren't focused on a particular domain but rather on building a flexible SDK for a number of domains. For each of those we had separate classes of transactors called *actions*. But then, in the

At root, we knew we had a problem, and they clearly had a solution. It was just that obvious.

course of doing code reviews, we kept finding bugs, which led us to conclude we were putting too much trust in the ability of our SDK users to write secure actions and deal with signatures and things of that nature.

That's when it dawned on us that we needed to stop resorting to custom solutions each time we ran across a problem and instead turn our attention to coming up with a common programming remedy.

At first, we thought we would write an abstraction layer and develop a programming language to go along with it. But then we found that the team at Elevance had already developed a top-down approach to address just such issues. So, at root, we knew we had a problem, and they clearly had the solution. It was just that obvious.

CF What does your architecture look like now that you've put it all together?

SK There are a number of components, but let's start with what we call our Abstract Ledger Model, which is an implementable specification for running DAML applications on all kinds of ledgers—by which I mean distributed ledgers and traditional databases and graph databases and event streams and even things we haven't thought up yet. Obviously, we have our own distributed-ledger platform, but that's also complemented by implementations for other persistence layers like PostgreSQL and VMware.

We also have what we call the DAML engine. Whereas DAML is our contract language, which you can think of as being built to provide for safety in multiparty workflows, the DAML engine is what provides for the interpretation of contracts written in that language. It also ensures that the

authorized commands that act on the current state of the distributed ledger result in a new state that's consistent with the contract semantics written in DAML. All of these layers sit within what we call our ledger server, which is a swappable component that implements the Abstract Ledger Model along with a runtime environment for DAML programs.

Other ledger providers now have implemented the same DAML abstraction and APIs using an integration kit that we've open sourced. This allows users to choose different blockchains depending on their own requirements and vendor preferences.

On top of all that is an application and integration framework that employs an event-handler model. This means that applications can react to events from the ledger while also submitting commands to it. Basically, that framework provides all the properties required to support caching, system restarts, optimizations, and the like. It attends to these sorts of issues so the developer can focus on writing the business logic. This is similar to the experience of using the serverless or lambda functions available from AWS [Amazon Web Services], GCP [Google Cloud Platform], and Microsoft Azure.

CF That's impressive! Anything else?

SK We also have an SDK, which isn't part of the platform architecture itself, but it does come with an embedded distributed-ledger simulator, as well as a testing language that provides a full semantic model of a distributed ledger—meaning that if it will run on your local machine, it will run on the platform as well. That includes analysis tools that leverage the structure of the language to check

for common mistakes and entitlement logic. From the contracts you write, it derives who, from among all the different participants on the network, should be entitled to see this particular data. All the entitlement logic is driven by the workflows, so that you, the developer, don't have to specify any of that yourself.

CF How did you manage that?

SK That actually comes from the structure of the language. As you specify your business logic, you specify the workflows. Then, with reference to the function signature, you can see all the network participants that will be affected by this contract or might come to be affected later on. This then allows you to analyze that for all future evolutions of the contract.

CF Cool! It sounds like you ended up with something that could prove to be quite powerful. Now tell me about some of the more interesting engineering lessons you learned along the way.

SK From an engineering-management perspective, there has been a lot for me to learn. I came to this from being a cryptographer who was working on cutting-edge zero-knowledge proofs and SNARKs [succinct non-interactive arguments of knowledge], so my intent was just to keep on using all those cool tools when we founded the company. Well, just as the token model fell short of our expectations, cutting-edge crypto also wasn't such a great fit. That was one lesson.

Another lesson is that, while I'd never written in a functional programming language before, I'm now a complete convert to both functional programming and strongly typed languages. I've come to believe that the

industry as a whole needs to evolve toward defensive programming techniques. Functional programming and strong types are among the best tools available for that.

I also learned that you can never be wedded to the tools you have or the code you've built. A year after launching this company, we had already completely scrapped all the code we had written and left behind most of the tools we had been using. Maybe we should have noticed earlier that we could manage all states on the ledger itself, but the important thing is that we ultimately *did* discover that. And that was largely because we listened to our customers to find out what their real problems were. This is how we found we were going in the wrong direction. Mind you, we didn't do *exactly* what the customers told us we should do to resolve those problems, since that would have put us completely off track as well.

I've also learned that you need to put careful consideration into every single tradeoff. I'll give you an example: Immutability of a blockchain makes resolution and recovery from data corruption a nontrivial problem. Accenture came out with a paper promoting mutable blockchains and was ridiculed for it. Having come from the blockchain community, I understand the reasons for the ridicule, but the paper itself was actually pretty balanced. It's just that the blockchain true believers weren't really capable of reading it in an unbiased way. I've learned you can't afford to be doctrinaire like that, since you need to be free to take an analytical approach to every single significant tradeoff that comes your way.

CF If this model you've built for distributed ledgers and the formal methods you use to define them prove to be widely

adopted, what sorts of changes do you expect will follow? I ask since one of the things you said you hope to accomplish is to open up innovation throughout the industry.

SK I think we'll see the same kind of Cambrian explosion we witnessed in the web world once we started using mutualized infrastructure in public clouds and frameworks. An example I often cite is that I had absolutely no knowledge of web development when I started a Bitcoin

brokerage in Israel some years ago, since I was very much a low-level assembler and crypto kind of guy at the time. But then it took only three weeks for me to learn enough Ruby on Rails and Heroku to push out the first version of a management system for that brokerage. And that's because I had to think only about the models, the views, and the controllers—which is to say only the business processes. The hardest part, of course, had to do with building a secure wallet, but that was my expertise.

Anyway, moving on to today's banking world, if we look at all the networks of financial institutions that currently have

mutualized workflows, you won't find any frameworks, abstraction layers, or mutualization of infrastructure. There's nothing that people with good ideas can use to

Related articles

➡ Bitcoin's Academic Pedigree
The concept of cryptocurrencies is built from forgotten ideas in research literature. Arvind Narayanan and Jeremy Clark
<https://queue.acm.org/detail.cfm?id=3136559>

➡ Research for Practice: Cryptocurrencies, Blockchains, and Smart Contracts
Expert-curated Guides to the Best of CS Research
Arvind Narayanan and Andrew Miller
<https://queue.acm.org/detail.cfm?id=3043967>

➡ Bitcoin's Underlying Incentives
The unseen economic forces that govern the Bitcoin protocol
Yonatan Sompolinsky and Aviv Zohar
<https://queue.acm.org/detail.cfm?id=3168362>

rapidly roll out new systems and then iterate on them or upgrade them. That means if someone comes up with financial innovations that might be used, for example, in the health-care domain to reduce payment counterparty risk, there's just no easy way right now to deploy that.

But should we find ourselves in a world where every large enterprise is part of one of these distributed networks that does mutualize infrastructure, then people would be able to iterate very quickly on these sorts of ideas. Basically, they would be able to propose new products and deploy them as smart contracts. The impact of that could prove especially significant in some of the large industries that so far have been slow to adopt innovations. I think the potential there could be huge.

Copyright © 2019 held by owner/author. Publication rights licensed to ACM.