Transcript

My name is Stephen Diehl, I'm the CTO of a company called Adjoint Inc. Normally the presentations I give are a bit more technical and involve quite a bit more equations and rigour, but I'm going to try and speak today in the pseudocode known as English. [laughter] I'm going to talk about smart contracts, but rather than delve into the deep formal methods that I work on every day, I thought I would give a high level review about what I wish I knew about smart contracts before I got into this space.

The goals for my talk today are going to be what should you as a non-technical entrepreneur know about the smart contract space, and what can I as a technical entrepreneur impart to you to help you on your path toward building your successful smart contract business. In particular, what are the technical threats to your business down the road, and what should you be looking at for the next generation of this space.

Terminology

Smart contracts.

- Many things to many people.
- Generally **not smart**, and **not contracts**.

Let's start with some terminology. Smart contracts, they're completely unambiguous... Of course not; they are many things to many people, but generally, they're usually not smart and they're usually not contracts. Smart contracts are effectively programs that run on blockchain, in most definitions, but generally they're quite simple cases of logic and generally they're not legal contracts in the legal sense.

Adjoint Inc.

1. **Settlement Networks**

- Creating executable forms of industry standard contracts (ISDA, EFET) for derivatives, swaps, options on mutually distributed ledgers.
- Network creation for consortiums that trade OTC contracts in commmodities and energy markets.

2. **Infrastructure**

- Laying foundations for 3rd and 4th generation blockchains.
- Verifiable computing
- Formal methods

What does Adjoint Inc. do? I'm the CTO of this company, we're here in London. The boring side of our business is we create settlement networks for creating executable forms of industry standard contracts, like ISDA and EFET agreements, for modelling the structure of financial products, such as derivatives, swaps, options on mutually distributed ledgers. I don't work in the public chain database at Adjoint; I work on private server networks between financial institutions, such as. Primarily we look at modelling executable forms of OTC derivatives contracts, and I'm particularly focused on taking a description of the semantics that involve the temporary rights and obligations of counterparties that are party to a derivatives trade, modelling that as code, and then putting that on a distributed database so that we can have more efficient settlement systems.

The more interesting part of our business is that we also spend quite a bit of time doing research and development on what I would call third and fourth generation blockchains. I do a bunch of research and development on verifiable computing and formal methods, things like zk-SNARKs and reasoning about the semantics of contracts formally.

My Initial Assumptions

- Consumers would want to know what their contracts do and if they behave correctly.

1. Most smart contracts are not realized, and exist only in a whitepaper.
2. Most smart contracts that are running are proof of concepts.
3. Most smart contract developers are fine with their being huge flaws in their contracts because their incentives aren't aligned.

The assumptions that I made when I was starting my business, the core assumption was consumers would want to know what their smart contracts do and if they behave correctly. That turned out to be false, for a variety of reasons. Firstly, most smart contracts are not actually realised. The ones that are running are generally proof of concepts, where if they run correctly or don't run correctly, nobody really cares. Thirdly, most of the developers that are writing the smart contracts in this space are contracts, because generally they're not economically invested in the long-term success of that contract. So we end up with this space, or this cesspool as I call it, of contracts that exist on the public chain, of which maybe there are five or so running successfully that are not trivial token registries.

Don't fall into Social Astronautics

When great thinkers think about problems, they start to see patterns. They look at the problem of people sending each other word-processor files, and then they look at the problem of people sending each other spreadsheets, and they realize that there's a general pattern: sending files.

When you go too far up, abstraction-wise, you run out of oxygen. Sometimes smart thinkers just don't know when to stop, and they create these absurd, all-encompassing, high-level pictures of the universe that are all good and fine, but don't actually mean anything at all.

Some Hard Truths

The entire space is very early. Think 1990s era internet.

There are shockingly few smart contracts that work.

Many good ideas are simply unrealizable as smart contracts for technical reasons.

Do your research before jumping head first into a project.

Let's go into the hard truth about this space a bit. This space is very early: think kind of 1990s era Internet. There's a lot of promising ideas that are emerging, but I would say there's a lot of hype and there's a lot of irrational thinking, so we're very much in the kind of ~Web bang yahoo~ era of the smart contract space.

There are shockingly few contracts that actually work, there are maybe about five in the world. There are a lot of token registries, and that's about it. The [~4, 22:02] of the world are largely spectacular failures, because a lot of engineers haven't put the engineering effort into building robust smart contracts.

There are many really, really good ideas for smart contracts that are simply not realisable for technical reasons. The primary thesis of this talk would be to think and do due diligence on the technology before you invest large amount of resources in building these contracts. That's kind of the dark side of this talk, now let's talk about the light side, about what we can do about those things.

Do your research before jumping head first into a project, make sure that what you want to do is actually realisable on the public chain networks, the private chain networks, and the contracts that you're trying to model, you have it clear in your mind about what they should do before you start to execute and code.

What Are We Talking About

- A piece of logic that coordinates an agreement between counterparties.
- Run on a network with certain properties:

1. **Identity** - Individuals and parties have an address or unique identifier.
2. **Persistence** - Data can be written and shared.
3. **Non-repudiation** - Parties cannot dispute the existence of transactions.
4. **Assets** (optional) - The exchange of value.

Talking about smart contracts, let's go back to first principles and talk about what we mean by first principles. I need a piece of logic that coordinates an agreement between counterparties, it is run on a network with certain properties baked into the network itself. One is identity: individuals and parties have a unique address that can be used to transfer data and transact with each other. Create persistence: data is distributed across the network and can be shared if needed to by contracts. Non-repudiation: parties cannot dispute the existence of transactions on the network and they cannot alter counterparties' data. Assets as well: on the public chain networks quite often we have an ambient cryptocurrency, and this is useful for some contracts but it is optional. You can derive assets from having data, because assets are basically just registries of value.

- Smart contracts are a not a new idea.
- Distributed databases have evolved recently.
- The on-chain logic can be used to model human workflows (i.e. contracts):

  Smart are contracts are a programmatic description of a sequence of touchpoints that act to synchronize data and reach an agreement on some time varying set of rights and obligations between parties.

Smart contracts are not really new idea. What is new, however, with blockchain is that we have distributed databases that have new properties. They're largely these global incorruptible sources of truth that we can build computational logic into, that we can use to mediate transactions and human workflows on smart contracts. I claim that a smart contract is a programmatic description of a sequence of touchpoints that act to synchronise data, and reach an agreement on some time varying set of rights and obligations between counterparties. That's one definition of a smart contract, and that's about as good as you can get.

The Issues with 2nd Generation

- Lack of talent.
- Token value creates perverse incentives and scams.
- Current languages are too low-level and enormously difficult to reason about.
- Need richer type systems in our languages.
- Provably fair computing over private data a technically quite difficult.

The issues with second generation, that I learned the hard way when trying to build smart contracts, is that technology is rather immature. This is why my company spends quite a bit of time working on what I would call third and fourth generation systems, because the chains that exist today are not going to exist forever. We're going to need to be able to write more complex logic on chain, we need to be able to have more languages that we can express richer, tighter ideas on chain. Right now if you have ever written a smart contract, I think you'll find that you're working at a level that's far, far below the problem domain that you've set out to work on; you're working at the level of bit shifting and hashing, when you should be working at the level of "~What does my counterparty need transacting about~?"

There's a lack of talent in this space, the token value creates perverse incentives and scams. The current languages we have are too low-level and they're enormously difficult to reason about. Solidity, I claim, makes PHP look like a work of genius. [laughs] We need richer type systems for reasoning about the behaviour of our languages [~3, 26:24] ~before they're run~, and we need to be able to have [~3], we need to be able to transact on ledger about off-ledger data [~3] ~privacy constraints and amass economic details of our trades~, because they simply can't comply with existing legislation and data governance laws or even just economic interests of the firms we transact with, if everything is stored in the clear on this [~2]. That's a big problem, and I think this problem is not addressed particularly well in the current generation, but [~5].

Reason

Can we ask our "digital layers" questions about the contracts before we execute them?

- Will P always happen?
- Will P never happen?
- Is P is true for all states?
- Will P eventually be true?
- If P ever becomes true, then will at some point afterwards Q must be true?
- If at some point P becomes true and then will it stay true?

Asking these seemingly simple questions of today's contracts is impossible in the general case.

I claim the biggest issue with smart contracts is ~the lack of passage of reason about the code we invent~. When I say reason about, I mean to ask questions. If we are going to represent contracts which are agreements between people about a workload in time, we go to our lawyers to ask questions about what does this imply for my business, my personal interest, my bank account; we should be able to ask our digital lawyers, or our software, to analyse our contracts and ask much the same questions that we ask our lawyers. These are questions that are basically [~3, 27:41] will something P, a predicate, always happen? Will it never happen? Is it true for all possible states of the contract? Will it eventually be true? If P ever becomes true, then at some point will Q become true? If at some point P becomes true, then will it stay true? Anybody who has delved into the world of formal methods knows that this is a description about the high-level predicates in temporal logic. I claim the properties that we want to state about our contracts can be phrased as temporal logic problems.

Legal Counsel and Digital Counsel

- Does my contract terminate?
- Does my contract put capital at risk?
- Does my contract accept nonsensical inputs to data feeds (LIBOR)?
- Does my contract allow me to opt-out if my counterparty disappears?
- Does my contract comply with EU data residency laws?
- When does my contract have cashflow events?
- Does my contract allow me to safely alter terms with bilateral confirmation?
- Is the voting in my contract provably fair?
- Does my contract not allow a single party to empty it?

If we translate those into questions about contracts, it becomes quite natural: does my contract ever terminate? This is a question that is shockingly hard to ask a giant pile of Solidity code. Does my contract put capital at risk? Does my contract accept nonsensical inputs to data feeds? Say if the LIBOR data is nonsensical, will my contract [~3, 28:35]? Does my contract allow me to opt-out if my counterparty disappears? Does my contract comply with EU data residency laws? When does my contract have cash flow events? Does my contract allow me to safely alter terms with bilateral confirmation? This is something called [~1]: two counterparties agree that they should change the economic details of a trade. Can they do that if they both agree? Does the contract allow that? Does my contract allow provably fair voting, if it's say equity or something. Does my contract not allow a single party to empty it? This has been a point of some dispute about the current generation of Solidity contracts.

Verification

Verification at two levels:

- **Implementation-level**: The coherence of the underlying language's implementation in terms of it's static and dynamics of the language's semantics.
- **Logic-level**: The coherence of a specification to a contracts expected behavior and it's implementation.

Verification is an interactive process to debug your thinking. The programmer needs to exhaustively specify the expected behavior in all states.
Verification is the process of proving coherence between the implementation and the expected behavior.

The term that often gets thrown around in this space kind of haphazardly is that we need formal verification, and quite often that's where the statement usually ends: we need formal verification. What does that actually mean? There's a lot of people that are looking at this, and I claim a lot of these approaches get confused. There's two levels of formal verification, and one is at the implementation level. Do we know that the underlying platform itself is correct? Is the implementation of say the ~the agreement virtual machine~ a fair and faithful mapping [~3, 29:56]. Is it? There's no formal proof that it is, we just don't have [~6], and that's a problem. I think that's largely actually being addressed, there's a large body of work ~that have to formally verify that an implementation in a language is correct~, [~3] in the last 20 years.

I'd say the more interesting problem, and one that often gets confused with verification and implementation, is actually the verification at the logic level. The contracts that are specified in a language, which we say is 100% correct: how do I know that the intent of my contract matches the implementation? That's a more difficult problem, and I claim there is no magic bullet that's going to solve this problem. I claim the answer is we need richer semantics that are much more [~4, 30:50]. I claim that formal verification basically involves distilling your problem domain down into a small set of reasonable components which can [~4] and maintain proofs of their correctness and their composition.

Elements

Contracts should be a high-level description of the relations between time varying rights and obligations.
Composed from simpler building blocks modeling the semantics of financial products:

- Confirmation
- Agreement
- Offer
- Novation
- Termination
- Membership
- Ownership
- Triggers
- Permissions
- Rights

At Adjunct we work on a collection of such building blocks for ISDA and EFET agreements, that we think almost all of the terms of the contract can be distilled down into these 10 different components, and then we encode them as small building blocks [~3, 31:28] ~in time and with each other~ to give rise to the standard financial products. These are things like confirmations, agreements, offers, novations, terminations, memberships, ownerships, triggers, permissions and rights. These are all small control structures that have specific invariants that ~under composition~ give rise to a larger set of invariants that we can feed off to a model checker, and then we just add the temporal logic properties that we've discussed before, so effectively that's how you do formal verification. The details of that are non-trivial, and that's what I work on. I claim if you're interested in formally verifying your contracts, the first thing you need to use to start doing that is to write down what are the simplest rights and obligations to my counterparties that change in time, and distilling it down into a smaller set of components that you can reason about ~is the first step to~ asking those questions that we need to have answered.

Data Privacy

Can we use public networks to handle to economic details for complex business needs?
We need new transactions to allow us to compare off-ledger data in a way that is irrefutable, pubically verifiable and can't be forged.
Next generation systems can get quite far with the ability to prove:

- Prove preimages to hash functions.
- Prove equality of off-ledger data with non-disclosure.
- Prove inequality of off-ledger data.
- Prove numerical (greater than, less than) properties off off-ledger data.

The other big problem that I've spent a bit of time working on [~4, 32:30] is data privacy is a big problem on the current implementation of current ledger systems. We need to be able to have our contracts have public and private methods, and private methods are units of logic that counterparties can transact with each other about data that they don't share with each other. There's a lot of cryptographic machinery that allows this, and there's been a lot of work on this over the last 20 years. We can't do this in the current generation of our systems, we can't even say, "Prove to a counterparty that I have a preimage to a hash function, that I possess [~3] private key." I can't prove equality of off-ledger data. ~We might have a party that wants to share~, say, "Is the master agreement or the hash [~2] to prove to each other that we have the same master agreements?" Again, there's [~4] to these things [~5] integrate with our contracts, because our contracts then act on data that's not shared, in a way that the counterparties convince each other, without sharing with each other, that they have the same data. [~4] economic trades, things like inequality of off-ledger data, and also things like numerical comparisons. If you're building an auction for instance, you need to know if a bid [~3] counterparty [~6] counterparty on the other side, without sharing ~every bit of it~ because it's sensitive and we can't show that in public.

Take Aways

- Our smart contracting tools should tightly model the semantics of the workflows we want to model.
- We need libraries of verified contract components that maintain proofs under composition.
- Verification is an interactive, not an automatic process.
- We should be able to automatically reason about our contracts and ask business questions of them.

The takeaways I want to take away from the talk right now is that formal verification is not a non-interactive process. It's a process that largely involves the debugging of your own thinking about the problem, and to do that we need to rely on tooling that lets us reason about the contracts formally, and we need to distil our problem domains down into smaller components that we can reason about. For the next generation of our systems, we should be able to ask these kind of questions of our contracts, and to do so is not a problem that's intractable. We can reason about code, we can reason about implementations of human workflows, and we should be able to interrogate them for properties that are relevant to our businesses.

Adjoint : www.adjoint.io
Mattereum : wwww.internetofagreements.com

Companies are working on this, my company is working on this, if you're interested, Adjoint, and Mattereum, Vinay's company. Vinay is one of the people I think who gets the problem domain here and understands the current needs of this generation of smart contract developers, and is ~looking at building solutions~ that address the problems I just discussed. That's all I have. [applause]

[35:35]

[Q&A]

Vinay: One of my favourite quotes is Ted Nelson saying the Web is what we were trying to prevent. There was a long period when every time somebody started a Web startup, it would ~typically dictate~ [~4] ~they would implement it~ and they would raise a bunch of money to build out more Web infrastructure. What do you think the difference is between a world where the smart contract ecosystem continues to develop in this messy, ad hoc way without proper semantics, versus a world where we get standards bodies, [~5] and things are semantically correct and verifiable? What do those two futures look like?

Stephen: I don't think there's a dichotomy between the two. Every technology ends up evolving organically, and what we're seeing right now is probably the most organic stage, I think the second generation is going to raise the capital that will build the third and the fourth generations. So I don't see an issue there, although I do think there's not enough interest in standards and building more complex contracts, and that's probably the more concerning portion for me, but I think it's going to be resolved over time. I think we're still in the ~Web band era~ of this space. [laughs] That will change and will give rise to... [~4, 37:22] was actually a viable business model [~3].

Christopher: I think both worlds will carry on developing in parallel, and it will be a gradual... Gradually, the kind of Internet of Agreements world will conquer more and more territory as it becomes more and more tractable. That's partly to do with achieving agreements between the players, and making the business case that it actually makes sense. If you look at the world of logistics, logistics has been amassed for decades, and they still can't get agreement to share data in a sensible way. There's been a reinvention of that wheel 20-30 times, at which point it really has to do with business models, to make it economically interesting for the participants to adopt a

new standard, a new set of protocols, so that you actually move that sector into this kind of world, just as one example.

Question: You explained the issues with the current generation of blockchains. I'm wondering if, without necessarily building new blockchains, is it still possible to find solutions for the issues on the under current platforms. Can we do something with ledgers [~2, 39:08] on this platform, or will we have to build everything from scratch and build new stuff?

Christopher: [repeating the question] Can we build layers on the existing platforms to achieve the kind of vision that we have in common, or do we need to strip it all down and build the foundations from the bottom up?

Stephen: No, I think we can get quite a lot of bandwidth out of the second generation. I think there's a lot of things that can be built on the second generation [~3, 39:55] ~basically global distributed stores that can model simple logic. I think there's a lot of problems that can be modelled in these systems~. I think if you end up building complex contracts, you'll hit a wall quite quickly, as a lot of firms actually have. But just the ability to even transact and execute small pieces of logic with counterparties is actually quite useful ~in and of itself and~ [~4].

Question: You talk about the standardisation of logic and semantics and different ontologies and things like that. Outside of the Web, those things have been produced ~sort of parallel~ with one another. In the future, do you see these things sharding off and being created in enclaves, or do you think that this will be standardised across one broad system?

Christopher: I can't speak for the logic, but certainly in terms of semantic standards you have lots of communities that are not aware of the existence of standards that are relevant to them, and you have ~a lot of "not invented here" syndrome~. To give you a specific example, the European Food Standards Agency has a fantastic messaging protocol standard: it's old-fashioned, it's XML with codes, but is really, really detailed, very rigorous and very well thought out. In the meantime, [~3, 41:44], the people behind the [~2] standard are reinventing the wheel and developing a new lab standard which covers exactly the same space – that's going on all the time. So the extent to which you can drive people to use the same standards, I think it's exactly the same phenomenon as you have with human languages. I don't know how many nationalities there are in this room, but we all speak English, and that's really driven by economics. We all speak English because that's where the business is, that's where the trade is, that's where the jobs are. The same will apply to semantic standards, if there is a business case that drives people to talk the same language, and then everybody will adopt it; otherwise they won't.

Scott: Just to follow up on that, a business case for adopting standards: both of you are working on work that has a very long history in computer science and is actually quite understood by a very knowledgeable set of folks that have broke their pick on these problems, learnt the errors, made all the mistakes. So what is it that we need to do to incentivise those people to join the conversation this time around? Because what's happening is they're not part of the conversation, this isn't actually having an impact. You have people running off on the blockchain, creating the same mess that was created over the last three decades by people who didn't actually pay attention to what had happened before.

Christopher: I have two answers to that, one is you are the person who has got an answer to the incentivisation problem. Secondly, there is something which some of my colleagues have pointed out which I think is quite interesting, is that blockchain, if you look at it not as a technology but as a phenomenon, as a social phenomenon, it is making people get around the table and saying, "Right, how can we use blockchain to solve X?" and then they realise, "Oh, but we need to have contracts to do that. Oh, but we need to do semantics to do that. Oh, and we probably need some logic to do that," etc. It's almost as though blockchain as a social phenomenon, as a bubble, whatever you want to call it, is driving people together, and that may result in some interesting side effects that may overcome some of the long-standing boundaries. Because even if it's completely false to say there will be one blockchain that will

rule them all and we will all read and write for that, etc., that some people go around saying... It doesn't matter: you've got people around the room saying, "How are we going to not lose out to this new world?" and then you explain to them the consequences and what they have to do, and sometimes they do it.

Question: Question for Stephen. The Parity multi-sig wallet hack, the latest decent-sized hack: can you tell us anything about that, and also ~the smart contract being~ the Wild West market [~4, 45:13]?

Stephen: The Parity multi-sig hack: Parity is an Ethereum client written in Rust, it had a major security flaw a while back that was actually patched quite quickly. The bug effectively came down to the transposition of a variable name in the implementation [~5], implementation of the wallet itself, then it exposed a flaw in the cryptographic machinery around signatures. That's an implementation-level logic flaw, it's something that fell out of the developers' carelessness, but it's also extremely difficult to write these clients. I think the fact that most of them are written in tools that have really, really horrible type systems, [~4] really difficult to reason about the correct behaviour to specify that behaviour, and to have the compiler check it is even more concerning. The Parity client is actually much better than average, and the fact that there's a flaw even in there is a bit concerning about the space. I mean, god only knows if there's tens of thousands of flaws in Parity that could be exploited by anyone or a nation state actor.

About the Wild West of contract verification, are you asking about people claiming to do verification solutions, or about the [~1, 47:00] of the problem?

Question: I guess both. What do you look for in choosing a smart contract long term?

Stephen: Somebody that understands your problem domain, of what you're trying to model, somebody who is preferring smaller contracts, smaller and dumber contracts. If you work in the publishing space, generally the lower surface area there is for the contract, ~your bugs can decrease~.

Christopher: It sounds like common sense to me.

Stephen: Indeed, you would think, but... I mean, a lot of it is infrastructure. A lot of smart contracts ~in the Ethereum Network~ are outsourced to like Russian or Ukrainian engineers who don't have a vested interest in ~producing coding~, [~3, 47:52] proof of concepts that are turned around, at which point they have no interest in proving that the contract is going to be useful or not exploitable. We need to have compilers and tools that can do the reasoning for us and reduce the problem down to a ~decision procedure that we can check automatically~. We don't currently have ~those in place~. We need to have those tools for the third and fourth generation.

Question: Currently contracts are written in natural language, because natural language is the language that's spoken by the business people who are interacting, and therefore it can convey the meaning of what they intend to agree on. Is there a danger... We already have natural language, and then ~the new interpretation of that sits~ almost two levels between... to get to what the intention is and what's written in the contract. If we're adding a third layer, which is software language, is there not a real danger that unless the whole world starts interacting, speaking and understanding software, that you lose meaning or you cannot possibly... How do you verify these contracts? Because surely you need to negotiate it in the first place in natural language, and is that not inefficient?

Christopher: I think you've put your finger on a very important point. But if you look at the history of civilisation, we've tended to hand over things to specialists over time. We handed over contracts largely to lawyers and specialists of contracts, and in a certain sense my expectation is that there will be a handing over to specialists who are able to look at the legal contract and look at the smart contract and check that correspondence, particularly if, as I said before, there is a good business case for that to happen. Yes, it's going to ~lead to complications~.

But when we went from wading across rivers and taking it based upon our physical strength to get across the river, to saying, “We’ll build a bridge and we’ll have a civil engineer who builds a bridge, and we will trust that civil engineer to actually make a bridge that works.” Well, there were a few hundred years when bridges were built, bridges would fall down, bridges would get washed away; these days bridges tend to stay in place, except in very exceptional circumstances. There’s a kind of learning curve there about how to build bridges, and we will eventually learn how to build smart contracts in a way that works for the people who want them, as opposed to wading across the river physically like we used to.

Stephen: I’ll add a little bit to the discussion about what’s the relationship between the smart contracts and natural language contracts. I claim most natural language contracts are not going to be ever put into an executable form, that there’s only a very, very small subset of contracts that [~4, 51:08] very simple workload that involves a simple set of touchpoints [~5]. Then we start talking about how do we take the ones that are amenable, to turn it into an executable form, and then how do we... Do we go from natural language into code, or do we go from code into natural language? There’s two schools of thought in the matter about that. [~7] whether we should extract natural language from a core calculus, which describes formally, mathematically, instead of formulas, about the [~2] rights and obligations of counterparties based on ~observable quantities~, natural language or restricted English from that code description, or do we take the code description and try to extract natural language from that?

The problem is that in both directions you have this loss in transformation, neither is actually sufficient to represent the other, because human language is naturally ambiguous, and code doesn’t capture all of the nuances of what human language can express. If you look at just a standard ISDA agreement, typically there’s enormous amounts of clauses and descriptions about states [~4, 52:32] simply not relevant, or they exist [~4] where is the jurisdiction, what’s the governing law, all these kind of things. I claim that really we’re only interested in a very, very small set of these contracts, and I don’t think we really need to be concerned about proving that the two are equal. I think there’s going to be contracts that exist purely in digital form, contracts that exist purely in natural language form, and that the intersection of those two is actually quite small.

Question: Maybe it will be necessary to get the contracts differently. One example is Creative Commons, where you have pictograms that represent some deeper relationships. I don’t know whether you are familiar with Creative Commons: you start from the pictograms which represent text, which is localised in as many languages as possible, ~unilaterally~ one version per language and that you can have an underlying core. So maybe we just need to look at the contracts differently and start to construct them a little bit differently.

Christopher: [~6, 54:04]. There’s an interesting history here: we’re all familiar with Internet languages like XML and HTML, and their historical origins are in SGML in the 80s, and that was constructed really to solve a problem that a big international company like Caterpillar had in producing its manuals for maintaining a Caterpillar bulldozer in 25 languages. They didn’t want to translate the English manual each time into 25 languages; they wanted to do semi-automatic translation, by having a very simplified version of their manual which could then be automatically mapped to different languages. That’s why these markup languages were originally created. So in some sense we’re going full circle. [laughs] So yeah, in principle, yes.

Question: You mentioned ~sensorics and ubiquitous~ computing in your talk. I’m curious if there’s a ~historically proven way to standardise the input that would be used~ in smart contracts, where hardware and software interaction has proven to be very successful and reliable in developing these standards.

Christopher: I think it’s a wide open field at the moment. We at TNO developed something called SAREF, which is an ontology for the Internet of Things, originally for domestic appliances but then extended, and that’s basically dealing with the problem you’re describing, which is you’ve got all these different sensors and they’re exposing their data in different formats, shapes and things, what are you measuring, in what unit, etc., and you need a standardised layer on top

that will make everybody be able to read that data in a similar manner, which means everybody needs to be able to be clear what the interpretation of that sensor really is, whether it's location, whether it's temperature, whether it's anything else. As we progress to build more and more sensors which are measuring more and more environmental factors, this is going to be an ongoing task, of standardising that.

Question: I wondered in terms of your research whether you've done any work looking at the rituals allowing the establishment of contracts or the evolution of them, the social rituals that are very established in the world, and what your views are on how they may evolve in this space.

Christopher: It sounds very interesting, but I know nothing about that.

Stephen: I just write code. I'm not a sociologist. [laughs]

Question: We've been thinking a lot about the future and how this could all go [~5, 57:30] if you imagine 20 years down the line that this ecosystem is at full maturity... I was wondering if you could give us your optimistic view of how this all goes through, and then, perhaps more interestingly, how you see it all going from your most pessimistic view?

Stephen: Pessimism I'm good at actually. What I would call the smart contract hellscape, where the entire world is moving financial infrastructure over to software that's not rigorously tested, it's amenable to large-scale hacks, we start seeing many, many cyber 9/11s everywhere, the banks' irrational exuberance about this technology, there's also a lot of loss... That's the negative side.

The more positive side is in 20 years we'll be eventually moving the entire world over to this global, heterogeneous, computational substrate, in which the entire world's computing infrastructure can be rented and leased, and is used to confirm and allow a global network to transact with each other across borders, it allows basically limitless free computing power for anybody to apply it in use, it leads to post-scarcity economics and [~5, 59:10]. [laughs]

Christopher: I think that's great. What can I add to that one? [laughs]

Ian: When we think about contracts, we normally ascribe the responsibility of the contracts to lawyers in the old world, and then we move forward into the new world and start thinking about smart contracts and verifiability and so forth and so on. Christopher, you brought up a lovely example about bridge building. We ascribe responsibility to the engineer, and we hope that the engineer has done a good job, which brings to mind the Canadian bridge that something like 100 years ago collapsed. In the aftermath of that disaster, with the help of Rudyard Kipling, they've created a ceremony where they presented to engineers the Iron Ring, which was worn on the pinkie of the dominant hand, so as you were working to draft your next bridge, your little Iron Ring would remind you, because it would keep bumping into things, that you were responsible – great ceremony. So my question is: with smart contracts, who wears the Iron Ring of responsibility for smart contracts?

Christopher: I think that goes back to something that Stephen has said, that we're in the very, very early stage, we're probably in the pre bridge collapse stage in the smart contract world, and when we get major collapses, then – somebody was talking about rituals – we might establish social processes, where we establish clarity and responsibility and chain of not just legal responsibility but in a certain sense cultural, social responsibility for those sort of things. Let's hope we get there before we get too many disasters. I think your point is entirely correct. I'm not sure how we can "engineer" that to happen without sufficient disasters to force people to change behaviour. In a certain sense, historically as a species, we tend to want a disaster before we actually sort a visible problem out.

Comment: Historically, the family of the workers were living beneath the bridge, that was the solution. [laughter]

Stephen: I would say that I think a lot of the onus lies on the engineers. I think that we as software engineering profession don't really have a good accreditation system around software development, we're kind of in the Wild West of... the entire profession, not just in smart contracts. I think if we designed software the way we designed bridges, I think we would take engineers out to the public square and shoot them, if they designed bridges the same way. So I think we as a software profession need to figure out who should be writing the contracts, how we should insure them and how [~4].