EPISODE 539

[INTRODUCTION]

[0:00:00.3] JM: Most applications today run on a cloud provider like AWS. They're built with a framework like Ruby on Rails and they use a set of APIs like Stripe or Twilio for middleware services. This is the era of Web 2.0. With decentralized systems, we're starting to get a feel for what Web 3.0 might feel like. The futuristic idea of Web 3.0 works off of the following idea: Instead of using a centralized service owned by a single company, you might purchase your computation and storage from a network of nodes. The nodes will be running peer-to-peer software that competes on price, and it may have a token network associated with immediating that economy of storage or computation or whatever kind of service you want from the peer-to-peer network.

Fabian Vogelsteller works on web3.js, a JavaScript Library for interfacing with the Ethereum blockchain. He also works on Mist, a browser for Ethereum. Fabian joins the show to discuss the difference between decentralized apps and centralized apps, and to explain why we need to build a bridge between those two worlds. It's actually pretty interesting to imagine a world where you have both decentralized technology and centralized technology. It's quite a buffet of options to be able to choose from for people who have been in web development for a while. I imagine it's completely overwhelming for people who are completely new to software engineering.

Nonetheless, let's get on with this interview with Fabian Vogelsteller.

[SPONSOR MESSAGE]

[0:01:48.5] JM: Software Engineering Daily is brought to you by Consensys. Do you think blockchain technology is only used for cryptocurrency? Think again. Consensys develops tools and infrastructure to enable a decentralized future built on Ethereum; the most advanced blockchain development platform.

Consensys has hundreds of Web3 developers that are building decentralized applications focusing on world changing ideas, like creating a system for self-sovereign identity, managing supply chains, developing a more efficient electricity provider, and much more. So, listeners, why continue to build the internet of today when you can build the internet of the future on the blockchain?

Consensys is actively hiring talented software developers to help build the decentralized web. Learn more about Consensys projects and open-source jobs at consensys.net/sedaily. That's C-O-N-S-E-N-S-Y-S.net/sedaily. Consensys.net/sedaily.

Thanks again, Consensys.

[INTERVIEW]

[0:03:05.1] JM: Fabian Vogelsteller is the developer of web3.js and Mist. Fabian, thanks for coming on Software Engineering Daily.

[0:03:12.6] FV: Yeah, thank you.

[0:03:13.8] JM: Before we dive too deep into the technologies that you've built, web3.js and Mist, these are some technologies in the Ethereum space, I want to talk about the term Web3 itself. Web3 is a term that is used to describe the next generation of the web, the decentralized web. But not every app is going to be decentralized. Some apps make sense to decentralize, but others makes sense to leave centralized.

Could you give some examples of applications that might make sense to leave centralized and contrast that with decentralized apps?

[0:03:52.4] FV: Sure. Decentralized apps are all the application – so what needs to be decentralized is always the core of trust. In the idea world of the Web3 vision is every component of your application is either running in a trustless environment or it's verifiable and it's done in such a way that nobody can manipulate the content, the files, the logic or anything like that. This is the idea.

An application where you don't need decentralization is probably everything where I don't know ownership, property or money is involved. I mean, you don't need to necessarily decentralize a game if you're playing it alone on your computer, right?

[0:04:34.8] JM: Or like Photoshop.

[0:04:36.7] FV: Or like Photoshop. Except you want to have some kind of collaborative Photoshop, but even there it really depends. Do you have some kind of core where multiple parties would need to trust? If you have that, then this is something which makes a lot of sense to decentralize and to actually run it in a trustless environment.

Today when we have these kind of situations, we always have a trusted middleman. We have an Airbnb, an eBay, a web server, a cloud hosting server, a government, a bank. Whatever is the middleman to basically make sure that two parties or multiple parties who might not trust each other can interact. In the future many of these things we can actually automate.

[0:05:19.5] JM: You're saying that today the main reason to decentralize your apps are to have an app that makes sense in a trustless environment. It's censorship resistant as well. Can you imagine a world where decentralized apps actually function more efficiently because there are so many peers in the peer-to-peer network?

[0:05:43.1] FV: There are different levels of decentralization. There's the decentralization of trust and there's a decentralization of computing or storage or file transport. So for example if you have a decentralized file storage, then the more people have that file, the faster you will download it. Currently we have the situation of there's one single server or multiple servers, the more people want to have something from that server, either computation or a file, the more that server has to work.

In a decentralized setting, the more half a piece or a part, the faster it will get for everybody else too. So when you talk about decentralization of computer, which is still to be worked on because it's not an easy task or when we talk about decentralization of file storage and transport of data, then decentralization actually increases scalability a lot. Especially the more people you have participating, actually the faster it should get technically.

When it's about decentralization of a trust or trusted executions, currently blockchains are the way to go, so the current status quo, let's say, and they have a limit to it. Because you want to make sure that the execution is run correctly and that nobody can manipulate. In current blockchain architectures, every computer is verifying these transactions. Everybody is verifying these computations. Therefore, there is a upper bound of what this computation can be. In this case it's kind of like we are running currently computer games like we did this in the 80s where they fit in a floppy disk, or where even a complex game with a lot of music and image material was done in such a way that either it fit in just a few kilobytes, or it was like very little material and compressed down that it actually fits in very little space, either one [inaudible 0:07:31.9] the floppy disks.

If you would take exactly the same computer game today, probably it would be a gigabyte large even if it shows almost exactly the same content. The reason is back then we had to very well architecture it, so it can run in this like few 640 kilobyte of RAM we had in our computers. Today, we have enough storage, enough space, enough computation power. We don't even have to worry anymore.

Currently in blockchains, we're at the commodore level. We are actually at the 386 level at this — A level of basically building our programs in a very optimal way to use as the least possible computing necessary, but still executing in this kind of blockchain environment. We gain the trust, but we don't overload the system and it doesn't cost us too much to use it. We are kind of like in this very early stage. I'm foreseeing that probably in like five years we will have super scalable blockchain and it kind of runs the same way like a cloud computing would run or even faster, and all these problems will be a thing of the past. We are still at the commodore level as of today.

[0:08:39.4] JM: Indeed. In the future, the decision to make an app decentralized or centralized will be as simple as an architectural decision. Just like people today choose between MongoDB and MySQL. In future, they'll be able to choose between decentralized and centralized. So if they are able to make that decision from an architectural point of view, what are the technologies that are in the stack of the decentralized app developer?

[0:09:13.9] FV: I mean I would actually go a bit further. I would say in the future, in a more far future, let's say, 5 to 10 years, we probably at the point where building things decentralized is the default and there would be no centralized execution or a centralized system necessary anymore. As it is technically the superior architecture if it's done right and if it's done scalable.

The tools today is, at least in the Web3 Ethereum kind of sense, the application is right now most likely a front and single page application. So it means it's a JavaScript application running in the frontend. The code is purely executed in the browser. It got delivered over a storage and file system like Swarm or IPFS in the planetary file system. So it's kind of like got retrieved in a decentralized manner and the advantage is that you know if you, for example, request this file and this file has to hash so and so, that you will get this file. So there's no man in the middle attack possible because you will hash the file on the end and you would check that the hash matches the one you actually expects to get so you know you got exactly the file which was linked wherever you got the link from, for example.

You download this in a decentralized manner knowing that you got the right data. You execute the JavaScript application solely in your browser or, for example, for that matter, in the Mist browser. Ideally this application is using only decentralized tools. For example, it could communicate over Whisper. This is a broadcasting protocol. It could use some kind of auto protocols, like telegram and what other encryptions, protocols. The idea in the Ethereum world is that it uses smart contracts as its core logic. You could imagine the smart contracts being the backend of the application and the application itself is running purely in the frontend.

Ideally though, you want to run only the minimal amount necessary in the backend because smart contracts, as I just said, are not that scalable and it costs you money to execute them. Reading is free. You can read as much as you want from the blockchain, but writing to the blockchain is what costs you money. There's also a lot of other things you can do and most likely in the next few months if you dig deeper into this, you would most likely think about state channels, payment channels, the Raiden Network to make payments. You will do a lot of things outside of the blockchain and you will use the blockchain only as an anchor and as a settlement layer. It's kind of like the security.

If you architecture this right and you can do this in such a way that your application is as secure as it would only run on the blockchain while at the same time having the scale and the throughput like a normal application. So it's really about architecturing this properly, and state channels is kind of the way to go.

[0:12:05.1] JM: I'd really like to focus on the technologies that you have built, although it's very nice to know that you're willing to just explain the whole stack. It's certainly something that's valuable. But given that you're an expert in the way that things are built in Ethereum, and you wrote Web3.js, or at least the first version of it and it's open-source and other people have contributed.

[0:12:26.7] FV: In fact, I wrote a second version of it. The initial version was done by Marek.

[0:12:31.7] JM: Oh, okay.

[0:12:32.0] FV: Marek actually worked on the initial version. I joined him them in 2015 and worked together with him and I actually now refactored it in the last year. I worked mainly myself on web3.js, refactored it to the 1.0 version. It is technically a complete refactoring, complete rewrite, which implements all of these applications, the APIs and the way how it should be done, the way how I actually envisioned it back then.

[0:13:01.9] JM: Okay.

[0:13:01.8] FV: So it's kind of like not backwards compatible, but it's a lot better than the previous version.

[0:13:07.1] JM: Well, all due credit to Marek. Let's talk about web3.js from your perspective. So I want to actually start with talking about the EVM, the Ethereum Virtual Machine, and let's work our way up the stack to web3.js. Because for people who don't know, web3.js is a way of interacting with the Ethereum blockchain via JavaScript. Can you start by giving an overview for how the EVM works? How is code processed on the EVM from a high level?

[0:13:39.5] FV: So the Ethereum Virtual Machine is kind of like the core little virtual machines computer which runs the code of the world computer, you could say. So every Ethereum node in the network is running the EVM. So, for example, if want to deploy a smart contract — So let's say I want to install a program on the Ethereum computer, I would send a transaction to the network with node 2 property, but with data, and this data would describe the smart contract and it initially constructs the parameters on the end. This is all described in byte code.

As the EVM itself is stack machine, only understands byte code. You would kind of like send this transaction in the network technically to nobody. The network would pick it up, install the smart contract, execute the constructor function with the parameters you gave it. Give it an address. The address is a deterministic address. It's based on your address hashed together with [inaudible 0:14:37.2] of your transaction. This is now your new program sitting on this address.

Depending on what kind of functions this program has, you can now start to interact with this. You can actually make transaction executing functions. You could also simply executing functions locally. This is basically just calling the contract and read from its current state or whatever functions you have made and whatever they return. They could return some calculation. They could return some internal variables from within the smart contract called the state, or you could actually just read the raw state yourself and so on.

You can basically interact with the smart contract reading and you can also send transaction to it. This is the same way you make a transaction, but in this case the tool is the smart contract's address. The function you call — So you put in the data field. The function you call plus the parameters, and this way you kind of like interact with this code. If your smart contract has a variable which has the one and now you make a transaction and it changes to 36, then the new variable for everybody readable after this transaction got mined is 36.

The EVM itself only understands spite code, like I said. So there is something called the ABI, the application binary interface. In order to talk to smart contracts on a blockchain, you have to first encode whatever your codes are into this ABI code. This is exactly where web3.js comes in place. It does all of that for you.

Web3.js is a library you use to interact with the smart contracts. So you basically just load this library into the application. In the ideal world, you're having a node locally running. So let's say you're using Mist. You're running your JavaScript application like a website in Mist. In this case you have a little object, an Ethereum provider just available and you can use this to call functions on the Ethereum node, and there are functions, for example, give them the current block number, give them the current account balances you have, or calling a contract or sending a transaction and so on.

Web3.js is kind of like using this connector point and there could be many other connector points. You could also connect over web server to remote nodes. You could also connect over a raw node IPC socket to a node siting on your computer and so on. Basically web3.js is using these kind of like transport layers to talk the node and takes away all of these complexity for you in a sense that it — You have basically the Web3 object and you can just call, "Hey, what other current accounts and what is the current block number." You can basically just call functions. They return a promise and they resolve with whatever they resolve this. If you want to talk to a smart contract, you need something called the JSON interface. The JSON interface describes — It's kind of like a JSON. It's an array with some objects in it and it describes how the smart contract functions are named and what are the input parameters.

Out of this little structure, web3.js is able to generate all these ABI calls correctly. So for example, if there's a function, hello world, and it receives — Set a new world as whatever, [inaudible 0:17:55.4] 256. Then to call this method you would have to take hello world, bracket, the type, [inaudible 0:18:05.6] 256 bracket. You would hash that name and this would be actually — The first four bytes of this hash would be the way how you call this specific function. Then you would have to append this [inaudible 0:18:19.0], which is then in 32 bytes with the integer encoded as hex. This is kind of like what web3.js is sending under the hood for you. It's as simple as calling that JavaScript function; hello world, and giving 1, 2, 3, 4, 5 or 6, 7, whatever number. Kind of web3.js is like taking away this complexity to dealing with this low level stuff and you interact with it kind of like it just would talk with JavaScript objects.

What it then does under the hood, it actually sends a transaction to the network. You have to wait until this actually gets mined, and in the new 1.0 version you have like convenience function like un-received, which means it's now mined and it's available or on confirmation and you can wait for a confirmation.

Confirmation means that you basically wait until your transaction is in blocks X after — Or like you're right now at block X after your transaction is mined. Let's say I want to wait for three confirmation, this means I want to wait at three blocks in the blockchain including mine 3 blocks before. Only then I'm sure that my block probably will not change anymore, because the tip of the blockchain has some certain probabilistic to being there and it's not final with the mining itself.

So you have this [inaudible 0:19:37.8] function, like on confirmation three, then you show the user, "Hey, it's mined," or you ask him to do something next or whatever. So you kind of have to deal with this delay. Things are not instant. This is probably in the future different with Casper and other consensus algorithms, but as of now you have to kind of deal with this probability in a way.

[SPONSOR MESSAGE]

[0:20:08.4] JM: Today's sponsor is Datadog, a cloud scale, monitoring and analytics platform. Datadog integrates with more than 200 technologies so you can gain deep visibility into every layer of your stack and any other data that you're interested in tracking as well. For example, you can use Datadog's Restful API to collect custom metrics from your favorite crypto data sources and analyze trends in Ethereum prices over time.

Start a 14-day free trial, and as a bonus, Datadog will send you a free t-shirt. You can go to softwareengineering daily.com/datadog to get that free t-shirt, and thank you to Datadog for being a continued sponsor. Get that free t-shirt at softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[0:21:06.0] JM: Okay. So you've given a really thorough overview from top to bottom, and that was a fantastic explanation. I just want to see if I can regurgitate it and make sure that I understand everything correctly that you said, or most of it at least, and give a re-briefing for some of the people who are less familiar with Ethereum. Ethereum is this big world computer and you have all the full nodes running replicas of the Ethereum world computer. They're all essentially the same copy of the address space that is all of the smart contracts and all of their storage across the Ethereum blockchain. They're all replicas of each other and they try to maintain consensus. They do maintain consensus with each other.

The smart contracts, the reason they're called smart contracts and not just programs is because smart contracts are tightly coupled to the idea of transactions and the idea of financial payment in return for computation. But these are basically programs. The reason you have to tightly couple the idea of payment for the computation is because you're paying the full nodes or you're paying the network to accomplish those transactions as a whole. So it is a contractual obligation on the part of the computers that are running these full nodes and it's a contractual obligation that you have to pay these full nodes to execute their computation. And the way that that payment is calculated is the opt codes, the EVM opt codes, the low level opt codes that all of these smart contracts compile down to so that there are prices for each of these opt codes and you can have reliable pricing, because all of these high level programs, these high level smart contracts break down to these EVM opt codes that all have these prices, and so it's predictable. That is how smart contracts work, except that to interface with these things, if you wanted to just interface with them from a raw level — I mean, it's similar to like you could interface with a backend Java application if you wanted to, but generally you're never going to do that. You're going to do it through a web interface. There are some — If you interact with Facebook, there's a bunch of backend Java server — I think Facebook has Java on the backend. Certainly Google does. So you interact with Google. You're going through a web browser and the browser hits some backend, maybe node application and the node application probably eventually hits a Java application.

You as a user, and even as many of the developers, are never actually hitting the backend API directly. You're hitting like — You may hit it directly, but you don't want to be — I mean, unless you're a backend programmer, you don't want to actually be programming it similarly unless you're a smart contract developer yourself. You don't want to be writing solidity code. You're only writing solidity code that compiles down to the EVM if you are a smart contract developer. But there are many people who would want to access these smart contracts. To access them, it's going to be much easier if you give them a JavaScript interface, which is what you have done with web3.js. Did I get everything right?

[0:24:23.8] FV: Yeah, that's kind of correct, minus the technical part. I find it very interesting how you call this is a smart contract, because you pay for it. The idea is actually — That's kind of correct. It's a very interesting way of seeing it. The name smart contract kind of came — It came from [inaudible 0:24:38.8], I think even in the 90s. This is this idea that you can have like automated execution between many parties.

So the reason why it's called smart contract is because it technically can have something like a digital contract and it executes whatever part the contract says in an automated fashion. So let's say a simple example, we have a buyer and a seller. They don't know each other. So the one wants to sell something, the other wants to buy something. You always have the problem with who is sending first, right? If the buyer sends the package first and the buyer never pays, that's a problem. If the buyer pays first and never gets the package. This is a problem.

In a smart contract you can built it in such a way that, for example, money is locked until really the seller got it and there's a way that the buyer kind of can force him to eventually release, click the button, "Yeah, I got it," and there are many ways of how you can do this. For example, both parties could put money at stake and they definitely want to get the money out. Plus, either receiving the package or getting the money. So you can resolve these kind of things in an automated fashion and therefore it's called smart contract.

It does not necessarily need to be a contract. It's [inaudible 0:25:49.9] simple code. It could be even a computer game. A smart contract could be a computer game. Even a guy who initially — I think this was very early in the beginning of [inaudible 0:25:59.5] called it the unreal engine to the EVM. That may obviously run very slow and would be a very expensive game to play, especially today. But you technically could do that. This could be just code programs doing stuff. But it can also involve ownership transfers.

A smart contract itself — And this is kind of interesting. The smart contract itself always knows who made the initial transaction. He has the address of the initial transaction, and it even has the address of the person who called him before. Because in smart contract world in the EVM, one smart contract can call another, and in one transaction you can have a call stack of up until 1024 calls. So one smart contract could technically call 1024 other smart contracts in the process and this kind of exactly is the reason that allows this very complex interaction.

Where the wallet or a smart contract of one person can interact with the insurance contract and another smart contract and the rental smart contract of a car, all in like one go and automating this whole thing of who gets what, when, this person fees or whatnot. This is what makes [inaudible 0:27:15.3] so interesting.

[0:27:16.7] JM: I certainly have coopted the term of smart contract from the high level definition to the low level definition, which is that you're entering a contractual relationship by executing this code and because you're paying for it and the person who is accepting your transaction as your customer is committing to executing the code. Of course, the contents of that code have a different level of contractual abstraction, I think is the clarification you are trying to make to me.

[0:27:51.2] FV: Yeah, exactly. I mean, there are many ways of — Obviously, the contract between me executing the Ethereum machine, then whatever the smart contract itself represents. Yeah, I mean the thing is kind of like a vending machine. Let's say there's a vending machine which gives you out cola bottles. It only works when you put in money. So the vending machine just listens to these coins coming in. Same here. The Ethereum computer kind of only works when you give it ether as a fuel, and then it executes. It executes atomically. So you really pay only for exactly what you use, for exactly the computations you are doing. Plus, there is a gas price that is variable. Like there is a gas price which you can set and which the miner can set, so the miner can say, "I want to at least have — whatever, 20 [inaudible 0:28:40.9] as a gas price," and I'm like willing to pay 21. Then I'm most likely to get mined.

If I say I only want to pay 10 [inaudible 0:28:49.2], then most likely my transactions gets never mined, because all the miners set it minimum to 20. The gas price, kind of the 20 [inaudible 0:28:57.0] which is 0.000002 to ether or whatever, this times the actual gas, which is just computation units you were talking about, the ones which each opt coin has a very specific number of gas. So your gas price times this is the actual fee you will pay in your transaction. You set the price.

So you as a user set the price. But if you set it to low and the miners won't accept it, then obviously your transactions gets never mined. It's like an incentive for you to set it high enough. For example, if you want to be the first transactions mines in the next block, probably you will set it very high. So there's this whole dynamic in the gas markets so to say as well. There's a few other details to it and so on.

[0:29:41.2] JM: The EVM does specify the price floor on executing a smart contract, right? Like by virtue of what the —

[0:29:50.2] FV: Yes, but this is a gas. So the number of gas is just a unit, like liter. For example, if I'm — Or gallons in the US. If I fill my tank, it's like 5 gallons or whatever, I don't know, and the price I pay is like — Let's say $1.50 per gallon. The actual amount I'm paying on the end is the $1.50 per gallon times how much gallon I tanked. Here we have kind of the price being the gas, which is this specific number. Like all the computations together maybe come to 20,000 gas, and the 20,000 gas times the gas price I set, like I want to pay, is whatever I pay. The miner can only either accept my transaction or don't. They will reorder me based on how much I pay. Obviously, normally a miner puts the one who buys [inaudible 0:30:38.4] gas price in the block directly and on the top, because he wants to get the transaction fees. If my gas price is way too low, I might not get in the block at all. It's only in like a few station anyway.

But maybe we can also move towards the Mist factor. What is Mist and how this plays a role here?

[0:30:59.6] JM: Oh, yeah. I guess go ahead and explain Mist. I did also want to talk about interacting between multiple smart contracts. Yes, let's approach Mist.

[0:31:07.1] FV: Okay. The Mist process is kind of the ideal —

[0:31:09.7] JM: Mist is a browser for decentralized apps.

[0:31:12.8] FV: Exactly. It's the ideal of what a decentralized browser can look like, and it's the first of its kind. There are many other approaches like MetaMask and Cipher browser, mobile browser — And Status, the mobile browser. Ether [inaudible 0:31:28.5], which is kind of like iFrame like browser. There are a few who want to basically make Ethereum the apps accessible. Mist is the only one who does it in the full fashion. It's a complete separate application, a browser in a way.

[0:31:46.5] JM: Let me just interrupt you. So traditional browsers, I'm using Chrome. Chrome can run JavaScript, and web3.js, which you wrote, lets me interact with the Ethereum blockchain through JavaScript. Why do I need an entirely new browser to interact with the Ethereum blockchain if I've got Chrome and I've got web3.js code running in Chrome?

[0:32:13.8] FV: Because the Mist browser is running in node. The Mist browser is actually Ubuntu. Now it's actually runs a full node. It runs the whole blockchain node and executes each single contract and each single transaction itself. Therefore, you are talking to the real Ethereum network. You're not talking to some kind of remote node where you have to trust that node, because this is the same as you talking to any web server. You have to trust that web server. The web server could tell you whatever.

So most of the other systems actually use remote nodes, because they have no real way of running a full node. This is good and bad and it has its advantages and disadvantages, but the main disadvantage of not running in node yourself is that you cannot really 100% trust the execution of the smart contract or the network data you're reading, because you just trust any third party.

The idea of Mist that you actually run your own node and therefore you know that whatever you're dealing with in Ethereum network is correct. In the coming Mist version, this would be a light client. So it would still be a node, but only executing the part where you're interested in. So your blockchain would be way lighter. It's just few 100 megabytes and gigabytes. The execution — The synchronization of the blockchain will be very fast.

Having a light client gives you the same security, because you still like download all the smart contracts and execute them locally and you check that everything is correct based on the hashes and the block headers. But you're not like dealing with the whole blockchain.

If you are working with nodes on other servers, the problem is you cannot trust them, and that might be in some cases completely okay. But imagine, for example, you have a smart contract which holds like $10 million or $100 million, and if this smart contract now says, “Hey, you just won whatever, and now send the transaction here and send like 10 ethers there and you get 50% off this 100 million just because whatever happened,” you won something or whatever.

If this you won is correct or not is very important, because if you sent here like 10 ether, you just lost $10,000. Based on you trusting this remote node telling you, “Yes, you won,” let’s say, right? You want to know that whatever you’re actually reading now from the blockchain is correct and [inaudible 0:34:36.7] tells you the truth. That’s right now and in the current web, actually the default everywhere, you’re trusting another web server which is controlled by another entity which could technically manipulate the results to his benefit and make you lose money or scam you or whatever.

If you’re really talking to a blockchain directly, if you run a light node, or if you run a full node, then you know that whatever you’re dealing with is correct. This is the key difference here. Plus, the Mist browser has another component. It’s not only executing the Ethereum blockchain, but it can also have all of these other decentralized systems built in. For example, you are able to browse the Swarm network. In the future you will also be able to browse the IPFS network. You can actually download your applications form the [inaudible 0:35:25.9] storage.

So it’s kind of like a combination of all of these decentralization tools and all of these new protocols which would like take probably forever to adopt. It will adopt it once that’s actually a critical mass. The Mist process is kind of like leading the way for, “Hey, what could be the future web?”

[SPONSOR MESSAGE]

[0:35:52.2] JM: If you are on call and you get paged at 2 a.m., are you sure you have all the data you need at your fingertips? Are you worried that you're going to be surprised by things that you missed, errors or even security vulnerabilities because you don't have the right visibility into your application? You shouldn't be worried. You have worked hard to build an amazing modern application for your customers. You’ve been worrying over the details and dotting every

I and crossing every T. You deserve an analytics tool that was built to those same standards, an analytics tool that will be there for you when you need it the most.

Sumo Logic is a cloud native machine data analytics service that helps you run and secure your modern application. If you are feeling the pain of managing your own log, event and performance metrics data, check out sumologic.com/sedaily.

Even if you have your tools already, it's worth checking out Sumo Logic and seeing if you can leverage your data even more effectively with real-time dashboards and monitoring and improved observability. To improve the uptime of your application and keep your day-to-day run time more secure, check out sumologic.com/sedaily for a free 30-day trial of Sumo Logic.

Find out how Sumo Logic can improve your productivity and your application observability whenever you run your applications. That's sumologic.com/sedaily.

Thank you to Sumo Logic for being a sponsor of software Engineering Daily.

[INTERVIEW CONTINUED]

[0:37:36.4] JM: How does that contrast with the MetaMask approach of having a Chrome extension? I believe the Chrome extension of MetaMask has a light client built in, right?

[0:37:48.4] FV: As far as I know right now, MetaMask is just connecting to INFURA, which is a remote node web server.

[0:37:55.2] JM: Oh, I see.

[0:37:56.2] FV: So the benefit of MetaMask is that you are controlling the private keys. It means you're assigning the transactions yourself in your extension. You don't like to give the access to your private keys to some different server and say, "Hey, please act in my name. By the way, I just locked in as person X or epsilon." You actually do all the signing of transaction on your computer. This is the benefit.

The disadvantage in a way is that you are actually dealing right now with a remote node. If they are able to implement the light client at some point probably. Right now, not. I mean, technically we could write a light client also in JavaScript. Even connect through the blockchain data over IPFS and whatnot and there are ways, and there are tries to actually make that happen.

Eventually you probably will have be able to have a MetaMask giving you the same trust in the blockchain than Mist, but then you still don't have all of the other new protocols and new different web decentralization protocols, which we can experiment within Mist. This is kind of the difference between Mist and all of the other.

The other give more an easy access, because with Mist you have to download the full application and in the past you still have to download the whole blockchain, which was actually horrible today. But you don't have the same possibilities, right? That's the main difference.

[0:39:18.7] JM: Let's focus back on web3.js, because that's what I really wanted to ask you some questions about. If I am a developer and I want to build an application that interacts with multiple smart contracts. Maybe I want to interact with a contract that — Let's say it's a lottery where I have a 50% chance of winning, and then every time I win that lottery I want to instantly to purchase KryptoKitties with that money that I won. Actually, maybe that's a terrible example of multiple smart contracts interacting. But give me an understanding of would I be using web3.js to build interactions between multiple smart contracts?

[0:40:01.5] FV: Totally. I mean, sure. I mean, if you want to talk to multiple smart contracts, first, you could talk to many. But if you want to do this in one transaction, you just send the transaction. The EVM would do all the rest.

For example — How it works is that I can basically talk to one smart contract and it can automate all kind of interaction with other smart contracts. This you would program within the smart contract itself.

[0:40:25.9] JM: You would build a new smart contract in order to facilitate that transaction.

[0:40:30.8] FV: Yeah. I mean, you mostly will build it in a generic way that like whatever. Like I get this, then do this. I mean, you could also build this directly in your application. As soon as I see money comes in on my address, then I send out a transaction to this other smart contract. Whatever you build inside your smart contract, whatever like multiple smart contract interaction you build in the smart contract or outside in the application and let them now send new transactions, which the user probably have to confirm or enter his password or whatever. That's up to the application and whatever you're doing.

[0:41:08.5] JM: Does web3.js, does it feel like the right abstraction to be accessing the Ethereum blockchain, or when you look at it, does it feel like — Like we had JQuery for a long time, and then people decided, "Well, JQuery is probably not the right abstraction to be interacting with JavaScript." Does it feel that way with web3.js or does it feel like the right abstraction?

[0:41:32.6] FV: I mean, in terms of JQuery, without JQuery it wouldn't have React and other tools. JQuery was kind of like the first step of, "Hey, how can we interact with DOM elements in a nice way?" Now we can interact with them. Now how do I interact with the DOM in a more abstract way?

Web3.js is probably right now the JQuery of the Ethereum world, you could say. If there will be better and more complex tools build on top of that or based on — Built on these kind of principles, very likely. For sure even. You could technically even build an MVC-like framework like React, but interacting with smart contracts.

The thing is as of right now, if you're interacting with smart contract and you want to change the state. So you want to make a transaction, Ideally you let your user confirm that or unlock this. You don't really want to send a transaction just under the hood in an automated fashion, because it would cost you. This way you could empty your balance rather fast and you don't want to store your password in your browser, for example, or in-memory.

It's more a usability question of how you would do that. What you can also do, and this is a nice thing, you could do some kind of two-factor authentication so that the application is kind of like showing the state of the smart contract or contracts, like for example your wallet and your

CryptoKitty ownership and whatnot. Every time you want to send a transaction, it presents you a QR code. You pull out your phone. The phone just scans the QR code, which is basically the [inaudible 0:43:08.1] transaction you want to make and you send it off from your phone. The keys are stored in your phone, in secure element, for example, of your phone protected by your pin or your touch ID and you send it like from your phone off to the network and not within the application itself, because the application itself doesn't need to send it. Whoever sends it to the blockchain, it doesn't matter.

This way you can have like very nice two-factor authentication and kind of like different ways of creating security. So you can have an application being only like the display, let's say, and the actual sending off the smart contract transaction. It's like your phone or another device.

[0:43:49.4] JM: I know we're wrapping up. We're in this place where there's a lot of excitement around decentralized applications, but there are not very many decentralized applications that are in production that people are using on a regular basis. Although I recently used Gitcoin, and Gitcoin has been awesome. I think that constitutes a smart contract/decentralized application.

So what are the problems that need to be solved? What needs to happen in order for these things to really start burgeoning and being popular, or is it only a matter of adoption? Do we just need the insurance companies and the bankers to start saying, "Okay. We need to be deploying these smart contracts." Like what are the bottlenecks they need to be overcome for these things to start being widely used?

[0:44:37.6] FV: It's more a question of architecture and the software at its current stage. I would actually say that a few decentralized applications which are used in an extreme large scale and, for example, CryptoKitties is definitely a big one, even though it's just fun. Transferring and creating tokens, and ICO is actually the biggest decentralized application currently running.

As weird as it sounds, but this is a type of a decentralized application. I can have a bunch of people investing into projects and tracking the ownership and buying and selling the ownership. That's a decentralized application, and it has actually many interfaces. Technically, every ICO has its own smart contract, have its own interface and wallets can interact with that, so you have this compatibility exchanges can end directly with these tokens who have this compatibility.

What we don't have yet is like this one application where now we see, hey this one application has the decentralized background, and that's it. There are many in development and I think the biggest hurdle right now is not adoption, because we have too much distraction. We actually have too much adoption. The biggest hurdle right now is actually the architecture.

So many of the applications which were built over the last years, they figured out, "Okay. The way I just built it where everything runs on the blockchain is actually not the way to go, because it can't scale or it gets expensive for users or have like a [inaudible 0:46:02.2] problem." First, you have to ether to maybe interact with it. Right now, the focus is actually on how can I build the same principle, but scalable.

Right now people think a lot about how I build the system of blockchain. How can I build this in such a way that I use the blockchain only for the core piece, for the settlement and not for all the other logic? This is the idea of state channels and so on.

We have a lot going on on the blockchain. That's not just like people testing the applications. Actually, a lot of like transferring money, anticipating ICO, creating tokens, interactions between different tokens and smart contracts. There is not the one application I would say to show. It's more like this kind of idea. Like ICO, it's a concept and there are many applications using this concept. Probably maybe that's what decentralized applications are. It's not like this one application and this one backend. Maybe it's this concept which now suddenly can all interact with each other. Suddenly the tokens can interact with each other and smart contracts can interact with tokens and wallets can interact with tokens and exchanges and users and you can have, for example, automated funds which just burst the assets they have in an automated fashion. For example, the Prism project from ShapeShift and so on and so on.

Right now, for example, a lot of people talk about and work on decentralized exchanges. How can we actually make this exchange risk factor and decentralize it in such a way that there's not one single point you can hack and suddenly get all the coins of everybody.

So I think we are still in the very — Like figuring out what can the technology and how can we right now make it work in a good way. We are still at this level. So if we have now all the insurances, like doing things in smart contract, probably not a good idea. The network is already full.

What, for example, would work very well, and this is something I'm working on right now is identity on the blockchain, because identity doesn't have too much write transactions. It depends on what you do. It can make basically people claim things about each other. So for example, I could receive a claim from an institution that they have my data and that they verified me and I can now participate in ICOs and having a KYC automated. I could receive claims either about my person or about my former work, for example, or a university degree and so on.

But I could also just claim things about — A computer game could claim that I own this character or that I own this digital asset or I could have as an identity owning different digital assets based in tokens or non-fundable tokens and so on.

We are right now still in this figuring out — Basically, we are building these core little use cases and these kind of like pieces. Once we have enough pieces together we can build these more complex applications. While we are building these pieces, we also are running into the limits of the blockchain and then we have to think about how can we solve this. For example, the last three years it was mainly about building the tools and applications to develop a need to even work on something. This was a thing developers had to do, is first, "Okay. I want to build this cool thing. Oh, I need this tool for that. I need that tool." Then it's about blockchain security. Let's build good debuggers. Let's build formal verification tools. Let's build ways to do tests smart contracts and make sure that they are secure.

There's kind of like so many things happening at the same time. It's really hard to say, "Okay. Now, I want to build this decentralized application. I have the perfect stack for that." You have to probably, along the way, figure out a lot of things yourself and even building then the tools yourself. That might be different in 5 years though.

[0:49:48.2] JM: All right. Fabian Fogelsteller, thank you for giving so much wisdom and building so much technology for this space. I'm really excited to see how your projects evolve overtime, and I think you paint a great and aspirational picture of where we can get to with decentralized applications.

[0:50:07.4] FV: Thank you so much, Jeff.

[END OF INTERVIEW]

[0:50:12.4] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers because I talked to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many newer hungry software engineers who are looking to level up quickly and prove themselves, and to find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineering.com.

If you're listening to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. So send me an email at jeff@softwarengineeringdaily.com. Thank you.

[END]