Slide 1 – Minute 1

Hello, I'm… talking today about Askemos.

The name Name „Askemos“ originally meant „A schematic operating system“. Whereby „Operating System“ was used strictly as in Hoare's definition "*The purpose of an operating system is to schedule access to the resources of a system among a number of concurrent processes.*". That is there is no relationship to „Hardware“ at all. Language is alive, the term has moved so much that we don't mention „OS“ anymore.

Within „Askemos“ resources are the subject of trade. Access is controlled by contracts. The digital model is these days known by the name “smart contracts”. I'm accepting it now, even though I don't find this name a particular good choice.

Slide 3 – Minuten 2-8

First a short overview on theoretical work regarding digital modeling of contracts.
Interesting by the way: It seems to be true that an idea comes into the world whenever the time is ripe for it. No matter which person is eventually praised to have been the first one. I have listed the time since when the authors have been working on the subject behind the date of publication. Obviously at least four persons where working on the topic around the same time. Who knows whom else did so. I – for example – only learned about the works of Grigg and Szabo in 2012.

The term „smart Contract“ was coined by Nick Szabo.
Szabo's work mainly introduces the idea that the text in contract documents are the “legal equivalent” to program texts.
In the beginning Szabo appears tp equate “the contract” with the technical artifact. However being more precise we see that the latter only checks compliance with the agreed upon rules. Furthermore he does not yet distinguish between the contracts text and the process going on according to the contracted rules. (This difference is also visible in computer science. However there, too, it's often ignored in colloquial speech. Example: we often say “firefox is running” - while we would have to say “a process is running, which executes the firefox code”.)
Eventually Szabo excludes the idea that „smart contract“ may be subject to legal dispute[1].
The latter has prompted the critiques to say the model would only amount to „state machines with money“ and not qualify as real contracts.

The citric I'm citing is Ian Grigg. His concept, the “Ricardian Contract“, introduces the criterion that the same digital contract must be readable to humans and machines at the same time and should be usable as evidence in case of dispute. After all contracts become important “afterwards”. As long as both parties are consent with the state of affais, the “actual contract” is an underpart. Only once there is a dispute, parties will seek a arbitration. That's the moment when the need arises to convince a judge to find a party in violation and correct them.
However the Ricardian Contract as Grigg defines it is not a „smart contract“: it defines a payment instrument. Slightly generalizing we using the term here to mean: it defines unambiguous rules, which shall be known to all parties involved business deal. Rules the parties may need to proof to

1 http://szabo.best.vwh.net/contractlanguage.html “Certainly a modern court would consider the interpretation I have placed on it in our language to be unconscionable, therefore unenforceable.”

an arbiter or judge. To this end Grigg especially enforces immutability of the document. This is being done by using a cryptographic hash over the documents content as the identifier of the document. (BTW: in „Zokoo's triangle“ these are global unique, secure identifiers, which are not meaningful to humans.)

Askemos unites both these concepts. From “smart contract“ at keeps the equivalence of “code” (as in contract clauses, law etc.) and “code” (as in program code). From “Ricardian Contract“ it keeps the addressing method: our objects are identified by a self-verifying cryptographic hash of their content.
Originally the idea came to be by accent. In '93 the one author did research into operating systems. Capability based security was a hot topic at that time. The unsolved problem was that those systems start out with a distribution of capabilities over actors. Given this distribution exists and are correct, it is possible to proof claims about systems where access is controlled that way. However how comes the initial distribution to be? It is defined by the “creator” or “owner” of the system in question. The super user.
The other author – father of the former – asked one day what actually the subject of his sons research was. Since dad had no idea about computers, but quite some knowledge about philosophy, especially enlightenment. Hence it suggests itself to explain “operating systems” as the set of rules governing access to resources among actors. Actors here are humans and resources contract goods. The role of the super user finds its equivalent in the feudal owner of the whole state.
At this point data said words to the effect “As long as this problem is not solved, all those computers may be nice toys. However they are useless for the real business life.” At the same time he offered assistance to clarify how the issue was solved in society during the enlightenment period. Translating this into a digital model was the only task left for the son. I never did more. Especially I can't claim to have invented any concept at all. Neither did dad. All we did was to digest 350 years old concepts again.
The consequence of the enlightenment period is in short the constitutional state. Therein humans can trade rights. However the can do so only within boundaries. There are inalienable rights. The latter belong to a human by birth. No despot or judge can grant or revoke them. Precisely those rights are the precondition which transform the mere human into a responsible part (and party) within the society.
Askemos is of course not modeled after historic constitutional state, it is an idealized model. It starts out with social contract („Du Contrat Social ou Principes du Droit Politique“, J.J.Rousseau 1762) – the conceptual equivalent to a constitution; or for that matter in the virtual world often going by the name of a license founding community (e.g. GPL, BSD) or „social contract“ (Debian). This social contract is like the „Ricardian Contract“ no real contract among parties. It is a single sided declaration of rules which may be used a the foundation (or component thereof) for real contracts.

One more note on classification. In the context of models and software it is often all to easy to confuse the model with the implementation. I'm using “Askemos” to refer to the model and “BALL” (Byzantine Askemos Language Layer) when I mean the implementation as executable code.

Now let's dive into some aspects. (Given that we have only 25 minutes, that's going to be a little terse.)

Slide 4 – Minute 9-10

What's a contract actually? The BGB (German Civil Code) lacks a definition (as much as it lacks a definition of proprietorship). The “Rechtswörterbuch” (legal dictonary) defines it a “a legal transaction consisting of matching declarations of intent (offer and acceptance) of at least two persons. The latter pledged with respect to the former. (Rough translation mine.)
The means at least: whenever we want to tack the label “modeled contracts” onto any system, we have to make sure offer and acceptance are clearly recognizable. Furthermore it must be obvious to the user that a certain act is going to the interpreted as a declaration of intent. And finally: no automatism may ever bind an individual to a contract.

Recapping here: the “ricardian contract” or “social contract” is no contract in this sense. It is a single sided declaration, immutable declaration clearly defining some rules. But it does not bind a person. This contractual binding only happens when a real contract is closed with referring this declaration.

The second clause of the definition is less interesting wrt. the effort to model contracts. However it ensures one of the most important concepts in our kind of society: the freedom of contract. Every person has the right to arrange their business their way by means of contract. This part of the definition however, when taken into the context of digital models and “smart contracts” raises the question: Do software patents lead to a clash with the freedom of contracts? If so, which one is the higher good? – But that's just a side note.

Slide 5 – Minute 11

BALL. However is a software. Written mostly for the purpose to show that it is possible at all to create a working and usable application under the restrictions of the Askemos concept. (Precisely that was doubted by literally every computer expert I talked to in the 90th.)

To this end we define a “virtual machine”. (Again in the traditional sense; not a virtual model of a real hardware.) BALL goes back to the tradition of LISP and XML here. This choice is by no means compulsive. It's been chosen because LISP, especially the Scheme variety is beyond doubt not “defined by implementation”. Furthermore the mapping from the textual representation to the corresponding abstract syntax tree is especially obvious and in the case of XML precisely defined by means of serialization into “canonical XML”.

Everything else about it is more or less by accident the way it is. From 2003-2010 there where several bachelor and master students working with the system creating prototype applications. Those have wishes and requirements for the system to be practically usable. And those things where implemented.

Slide 6 – Minute 12-14

The Askemos Virtual Machine must first and foremost be able to model our problem space. Actors communicating by way of message passing seem quite right. Contracts are closed by actors (natural or legal persons). To this end actors communicate by passing messages among each other. Each actor is modeled by a “state machine”.

Our machine must deliver the same results for each party of a contract. This leads itself to map the main memory addresses of the von-Neumann machine to Hashcodes. That's what we do.

At least for immutable objects this works pretty well.

Item become/update:

However actors, that is processes – in the given context modeling the state/compliance with the contracted rules at any given time – do change their internal state. If the address was to change each time, we could not easily implement a conventional programming language on it. (Which we desire to do.)

For this reason those actors keep their initial address, even though their internal state changes. We'll need a different way to proof their correctness. To be presented in a minute.

Items “new”, “link” and “read/write”:

Each operation (transaction) my create fresh objects (new). Objekts, respectively their identifiers (HashID) may be bound to human meaningful names locally to the actor (link). Furthermore messages may be sent to other actors (read/write).

Item “grant/revoke”:

Those capabilities I mentioned before are found in this machine too. However slightly different than in traditional systems. The latter used to use opaque identifiers which can't be made up by an (malicious) application. That's hard in a distributed context where all parties mutually distrust each other and the message transport beneath. Instead we use easily generated certificates. Their effect as a means of access control is again subject to a local proof by the BALL software.

So how do we model inalienable rights? Similar to “certificate pinning” we do so by not simply checking the existence of a valid chain of certificates. We only accept the outcome of such a check by double checking with local knowledge about the cert. Up to the point that we do not even require the certs to carry a cryptographic proof at all. Those certs are in turn “ricardian contracts” declaring the intended purpose to the holder. However we do check the structure of the certificate chain. We only allow the transfer of a **real subset** of the rights from one actor to another. We interpret each chain as being rooted in an actor and interpret “real subset” as a chain being longer and having a common prefix with the capabilities already owned by the actor granting a permission to another.

Item “replicates”:

Left is the question “How do we proof correctness or state?” (Be it the capability distribution or the internal state of applications.) Of course no party is inclined to trust the other party – or any third party – with that one. Therefore each party does the bookkeeping itself. Somewhat like doube entry bookkeeping.
Having two parties this can still easily lead to disagreement. Therefore we model what we do in the real world, whenever we want to be sure that the other party keeps their word. (After all bit patterns in memory are easily modified without leaving a trace. The are as good as evidence as a verbal utterance or human memory. Much unlike a writing cast in stone.) So what does a human do? They bring a witness to the deal.
The Askemos virtual machine keeps an identical copy of each actor which models a “smart” digital contract at each of the parties physical machine as well as on the machines of the witnesses. (We call those machines “notaries” in this context. Alluding to the role they play in the model.) Each change needs to be approved by those “notaries”. That is, we require “byzantine fault tolerance”.

Each single parties machine may fail at any time, even with malice. The 2/3rd majority will compensate.

Slide 7 – Minute 14-17

OK, that's it. There is nothing new going to come now anymore.
I'll simply try to explain the concept by example.

Imagine we don't want to use BALL. (After all who likes to work with LISP and XML?)
Instead we use "git". Everyone uses git already anyway. (Whereby "git" is again merely a symbol for "Merkle-Tree". That is saying "we store all our data in hash trees as git does.)

For eadch actor we create a git repository.
The first is the "social contract". This is a nice chance to make is ostensibly clear to everyone who gets physical access to our repositories under which circumstances access is actually allowed. If there is some remorse left behind one of the eyes looking at it, those shall be unable to invent a legal theory why they are allowed to access that data anyway.
The result is a HashID for a uniqeue device being owned by a certain person.

Next we define one actor (account) for each party – referencing that social contract as source.

Slide 8 – Minute 18-19

From now, both parties maintain both accounts. Better: each brings one more peer. Byzantine agreement allows a first failing party for at least four notaries. (See L.Lamport's proof.)

If we want to create a payment order for instance, we first need a payment instrument. That's easy to define the way Ian Grigg described it. Simply create an RC claiming one gram of gold, or whatever security you want.

Next send (e.g., vial email) a payment order to all notaries (parties and witnesses).

Each peer opens a branch for the account.
Broadcasts "I've seen a formally correct order at time X with hash Y."
Count signatures from other peers for the same par X,Y.
Broadcast "Hash of resulting state"
Count signatures for the result hash
Once there where enough: merge the branch into the trunk
Eventually send out resulting messages.

Slide 9 – Minute 20-21

Now lets look at some software to compare-

Bitcoin brought us the blockchain. A "new paradigm" it's being said to be. Since there are some attempts to implement “smart contracts” on top of this concept.

Just why wold one want to do so? I still don't understand that. The Merke-Tree, that is saying the part of the blockchain which may be replaced using git, that one I understand. By why do we need the proof-of-work and the lottery? Byzantine agreement gets the same job done. Just faster and without a global data structure.

More questions are left open:

- Is there really a contract being modeled? Including offer and acceptance? (It's not only German Law which requires this. Common law, despite many differences, requires according to my understanding the same.)
- Do those systems actually audit the activities and state transitions of those “smart contracts”? Or is this just an optional thing which one **could** do? (Something real users are certainly going to skip most of the time.)
- Repudiation? How are conflicts resolved? Nothing would be worse than leaving a machine the last say when it comes to deciding legality.
- Can the system model immutable rights? (If not, what's about my human rights?!!!??)

More interesting than the similarities are the differences here. Not differences in names but concepts.

I can't say much about Codius. Too little documentation I found.

OpenBazaar at least deploys Ricardian Contracts. But relies ony “moderators” instead of witnesses, which suggests that the audit may be incomplete. Again, I'm not competent enough to be sure.

OpenTransactions seems most similar to BALL. It too relies on the concept of notaries (up to the point of calling them by the same name). Those notaries come in “voting groups” much like the “replicates”-property of actors in BALL.

Slide 10 – Minute 22

Ethereum vs. BALL are worth a table.

The main difference in this group is the interface to the “global computer” and the format of proofs. BALL relies on Canonical XML, Ethereum uses a proprietary bytecode.

Slide 11 – Minute 23-24

The audit model to settle was already mentioned.
BALL follows the model that all parties do the bookkeeping. If there are too few parties: bring additional witnesses (notaries) which are going to be asked in case of dispute to determine the (legal) truth.

The model “blockchain” looks more like a public register. Like a cadastre. Useful for certain purposes, but often a bit of an overkill.

Let alone the question on whom one has to eventually relay to enforce the legal state of affairs. BALL requires 67% of a witnesses chosen by the user. Hopefully with care.
The blockchain approach randomly selects a single one from a pool. And this one needs to be backed up by 51% of the total Hashpower (which is another way of saying “those who invested the most money info computing gear”).

Troubling may also be who needs to have access to the contract/program text. BALL replicates it precisely to those choose witnesses. The blockchain again – if I did understand that correctly – needs to make it globally available..

Furthermore the blockchain is much slower. Bitcoins needs 10 minutes (per design). Ethereum „some time“. BALL completes transactions normally in less than a second.

Slide 12 – Minute 25

Eventually: those inalienable rights again. BALL looks at the certificate chain and restricts to known to be safe transfers of capabilities.

Ricardian Contracts connect fro the machine to the real worlds legal system.

Both missing at the right hand side.

Slide 13 – Minute 26

Thank you.