# Askemos - a distributed settlement

Jörg F. Wittenberger
softeyes
Erlenstr. 22
01097 Dresden, Germany
Joerg.Wittenberger@pobox.com

28 April 2002

**Abstract**

This paper presents Askemos[1], an autonomous, distributed operating system on top of peer to peer networks which significantly raises the level of abstraction in comparison with today's operating systems. Askemos addresses safe, secure and correct (forge proof) information processing while securing intellectual property in an innovative way.

Askemos defines a virtual machine on document level, which is defined in terms of abstract trees and pure functional transformation of them, both described in XML.

This virtual machine has no physical representation at any single machine. Instead it works distributed among independent components which appear as if they observed it. To achieve that effect, the participating machines compute the process steps of the virtual machine independent and vote among each other about the correct result.

To prevent illegal attacks, there exists no concept of unique resources like superuser rights or unique name spaces.

## Introduction

The digital age started out with several promises. Among the big bunch of curious to abstruse dreams are some more serious plans. One of this ideas is the "paperless Office". As a humorous definition says, it's "the one with the highest rebate at the local paper store". But these kind of jokes contain a grain of truth.

Media are used for storage of information since the invention of written language. As the information volume grew, the mass and cost of the media were reduced. From stone over clay to papyrus and parchment through nowadays paper.

A logical derivative would be to use our computers abilities accordingly: computers can not only store as much information as the paper equivalent of a medium sized forest, it is also able to process it.

But paper is still preferred! So: what is wrong with todays computers? – A short glimpse will show up: they are not reliable!

Even an acid bleached piece of paper will be readable in 20 years. Try that with digitaly stored information: Let alone physical damages, chances are, that the format of the information can not be read anymore in 20 years.

There are some examples at hand: Software evolves and data formats change. Sometimes the old software ceases to work for some reasons - if only because it was just upgraded. When a newer version does not read the old data and there is nobody to fix it, the data is lost. With a good portion of luck and if the word processors were made by the same company, you can import text from documents you wrote five years ago into new word processors. However you will already loose some formats. In the worst case, you won't be able to read anything, and loose all of your information.

This is even more likely to happen with 20 years old data.

Or think of a contract you kept on your computers hard drive quite a long time. One day you come home and your child tells you: "Look here what a cool game I've got. By the way, I had to make some space at the hard drive. Don't worry I removed only old files..." Should this even be possible?

Let us resume, how it works today - in a world based on sliced forests. Documents like passports and birth certificates are printed on paper. You need your birth certificate to get a passport. This birth certificate is your's, you are the owner. But there is a copy of it at the registry office. If you loose your copy, you get a replacement. For the passport, you have to leave a copy of your birth certificate and you might leave several more copies elsewhere. This secures the information against forgery or physical damage. One copy is nothing; the same information on many copies leads to a conclusion. There is

no 100% guarantee but a well grounded assumption, that nobody can forge or destroy all of these copies.

Another implication is, that a single copy is worth nothing. Try forgeing your birth certificate and ask the registry office to update their "backup". The clark at the registry office might be able to do so. But it's going to be hard to forge all traces.

Remember lots of old news about defaced web sites and stolen data bases and it should be obvious, why people don't trust computers concerning correctness.

But computers are not all that useless. When the paper based library of Alexandria burned down, most knowledge was lost as only a few copies where kept elsewhere. Computers can do better here utilizing distributed storage systems like freenet[2] or ocean store [3].

Askemos is an attempt to define a special purpose operating system, machine independent and distributed, with strong emphasis on trustworthiness. It shall fit those application domains, where todays computer infrastructure exhibits deficiencies: governmental use, free speech, long term knowledge management, trade and payment, law enforcement and similar purposes. These domains share a common point: information should - even in the event of physical damage - be safe, correct, and provable to such an extent that people finally find it suitable to gauge these processes. No attempt was made to invent something absolutely new, instead the working rules from the paper age have been adapted to the digital world.

Askemos must be safe and unbreakable. The author feels personally challenged to find a way to break it, but has at this point no idea, how that would be possible. That was also one reason to publish the sources of the System.

> "Security through obscurity does not work!"

## Threats and Requirements

Information resources and processes must withstand several threats to achive reliability and trustworthiness. Required are:

**Safety and Availability** Information can be kept permanently available by a massive backup in a distributed storage system like freenet. But preventing illegal and incidental manipulation requires strict rules to be followed when data is to be changed.

**Manipulation and loss must be prevented at all cost.**

**Security** Proprietary and private information has to be kept as a secure and confidential property of the owner. This very requirement forbids the existence of any administrative power, which can technically overrule users personal decisions.

**Liability** Accompanying the privacy requirement, a purely legal requirement already ensures use of illegally obtained information to be an illegal act. Keeping that rule effective requires a certain degree of liability and hence traceability.

Furthermore, trade and law enforcement depends on liability.

**Integrity** Quite basic, general, and usually fulfilled. Integrity means that no surprising changes can happen to data. Corresponding checks safeguard usually against transmission and storage errors. In Askemos they are extended from plain data checks to double check that the meta data properties, pertaining to liability, are intact.

This paper starts with the concept of the virtual machine of Askemos, it then goes to explain some detail about it and finally implementation decisions and remarks about the state of development of the current prototype are provided.

## Towards Reliability

The Askemos kernel provides denotation systems (minimum languages) for the three "axes" of information: data (visible, internal structure), association context (black box environment), and rights. The utmost invariant it assures is trustfulness.

The Askemos kernel is the communication infrastructure. As such it must inevitably be available and capable to denote all kinds of communication, which implies its utitily also for unpopular and even criminal use - an inherent, unavoidable property of all infrastructure. The requirement of being universally available and the fact that security through obscurity does not work anyway, where the primary reasons to publish the source code under the GNU public license[9].

Requirement specifications set for the kernel are the following:

- Trustworthiness shall be the ultimate design goal. All components are selected with the criterion of provability in mind. This criterion shall overrule practical wishes if they collide. Integrity is being assured using appropriate cryptographic algorithms and protocols. Neither shall assumptions be necessary about

honesty of owners or users nor is any level of security of network hardware assumed. (Unfortunately the security of the host machines is required in order to avoid a certain risk: data leakage.)

- The system reaches availability at any negotiable rate through redundant distribution.
- A protection scheme, which is not breakable in theory, secures privacy and information. Take over of host machines may not cause data leakage of any more than exactly the data those machines where used to support.
- No single point of failure shall be able to stop the system running.
- All processes and objects shall be "in system", there shall be no magic exceptions. Objects shall be complete and minimal (small is beautiful), i.e., in strict sense correctly abstracted.
- The system shall be implemented in a technology independent manner using widely deployed and open standards whenever available.

If an implementation does not follow these requirements, it shall be always called a bug and never a feature, regardless of the reason behind the deviation.

Requirements set for user level applications:

- intuitive, simple and understandable
- tight coupling between developers and users (synergy effect)
- modelled documents should exhibit a paper-like behavior
- direct manipulation of objects should be possible, especially their trading
- alternatives and migration paths should be part of all designs

Finally it should be noted that these requirements are merely a statement of intend. At the time of writing, there exist only a few actual applications.

## The Distributed Virtual Machine

The key concept of Askemos is the distributed virtual machine, which executes informational processes. This machine is defined in abstract concepts of documents and meta data handling.

The Askemos distributed virtual machine appears to work within the information space, an abstract virtual space where information is kept, no matter of media and physical location. Consider the example of a remote chess game. Two chess masters and an arbiter in a telephone conference. To stress the point even more, all might be required to just imagine the game board. There happens just one informational process, we understand it as one chess game, not as three. The electro chemical signals in the brains and phones are understood as projections of the process into a certain medium. Now add physical chess boards or a paper transcript, these are just more physical projections of the same idea into some data space.

The same principle applies to the Askemos virtual machine. This machine exists and executes it's processes in the information space. At physical level, several participating entities (which could be machines but also human brains) compute their individual idea of the process state in a somewhat synchronous fashion. These "individual ideas" are understood to be projections of the same abstract information, no matter, how different their encoding might be. For instance, one computer could keep it's projection of a picture in png format and another one as jpeg, it is still considered the same picture and the human brain will recognize that easily without keeping it in either form.

It was an error if a physical machine or human being would indicate a different process state than human understanding of the same process deems correct. This entities' projection is deemed wrong - at least from the observer's point of view. What if changes in the human understanding of the process itself produce a new formal specification? The old process is still executed, according to the old understanding and as such correct. But new theory is already in the information space and the Askemos virtual machine correctly executes both, the old and the new version, though in practice the old idea of the process will rarely be referred to.

The Askemos virtual machine can fulfill the above mentioned requirements (safety, security, liability, and integrity), as these properties can be formally specified. The availability of the information is still at risk if there where too few projections of it to synchronize. This, however, is a matter of statistics and can be adjusted to any desired level.

The following sections will present the definition of a minimal (and hopefully complete) set of aspects, also called axes, of the virtual machine. Care has been taken to keep formal specification and validation possible, based on sound and proven theory.

## Places in the information space

Many taxonomies have been proposed how to structure the perceptions used to express ideas, especially when it comes to programming. Leaving the latter aside (to see the wood from the trees) we find that at minimum an idea is a set of associations from properties to values. These associations are usually called pairs, arcs, or arrows. If such a set represents an idea, it can be identified, persist and - under certain circumstances - be forgotten. In terms of computer science this is a frame and the foundation basic building block of the Askemos virtual machine. A well known concept, not in need to be detailed here.

With respect to human experience and natural preference for three dimensional models - i.e., for the sake of end users, we call such a basic building block a "place" in Askemos. The distributed virtual machine ensures certain (a minimal set of) invariants about the properties of places. These invariants pertain to the requirements set out before.

Places have the properties of processes and objects in other operating systems. They receive messages. Upon reception they react with a possible property change (controlled by the virtual machine) and send messages out to other places. All in an atomic operation: the well known "process step". Places are autonomous: this is the only way to change the properties of a place.

A message has the same frame structure like a place, except that they do not persist and ca not really be identified: they are either present or not.

This latter choice of the model, even though it shares semantics with the programming language Erlang, the SOAP standard, and other similar models, might look somewhat arbitrary here.

Apart from the arguments given below, the author personally considers RPC a broken design. There are many communication processes and media in the real world: Nervous systems, attractants as used by ants and bees, electrical circuits, human voice, mailed letters. All of them are unidirectional and asynchronous - postmen do not wait for you to answer the letter they just brought. With asynchronous unidirectional messages, a broad set of reactions is possible at both sides. Both the communicating processes can continue in the event of failure. RPC was explicitly invented to hide the distribution of objects, and make calls appear as if they were local. It does a good job in that, but it aims at the wrong direction, because this just amounts to an arbitrary restriction.

First of all, the model can easily be mapped to the elements of the petri net theory[8], a quite capable theory for modeling dynamics. This allows to formally specify the dynamics of message exchange between the places, analyse them and proof certain properties e. g., for being dead lockānd life lock free. To do so, the data property of places take roles of places in petri nets, the state changes become transitions and messages content map to markings.

Second, the model computes the result of transactions in an atomic operation from just two parameters: the current state of the properties of the place and the properties of the message. Regardless of the programming language used to express the computation, the effect maps to a function evaluation. To the authors knowledge, pure functional programming, without side-effects, is the only way to make formal proofs of program correctness possible[7] page 175.

Third, the chosen model divides processes into process steps, which are visible to the underlying machine (agent). Chosen at a reasonable granularity (amount of real computation time needed for a single step), we derive a very convenient point and a restartable operation to validate and synchronize the projections (copies) of the same place with the agents at other physical machines and human brains etc. (see section about byzantine protocols), who also supports the place (maintain a copy). In case of human brains, one would be inclined to write "understand the place" rather than "support the place", but the author hesitates to use such term for programmed agents, hence humans brains better "support an idea" if they understand it.

This synchronization moment is furthermore useful to store the process state in several kinds of persistent memory. It is a well know fact that the larger and more reliable a storage system is, the longer takes it to store the information. Main memory is used to track each assignment of the processor – something infeasible to do at the hard disk. Let alone distributed, shared memory systems such as freenet. At the moment of synchronization a snapshot is taken. From that moment the process can already continue, while some more time might be needed the snapshot might need actually store the process state.

Finally when comparing to recent research about intrusion tolerant replication system [4], it becomes apparent (as intended) that the needed primitives map exactly to those provided by byzantine replication architectures.

## Topology and association context

For each place there is a globally unique object identifier. In the interest of liability, the identifier should be somehow a function of the content and meta data of the place, this way forgery would be impossible or at least detect- and traceable, as the documents leave their place, i.e., change identifier, whenever

the content changes. Such behavior appears to be imperfect for processes, as they become practically inaddressable. Hence, it was decided to keep such an invariant only for authentic documents in the meaning of constant, unforgeable facts. In practice the object identifier happens to be a cryptographic checksum from those properties of the place, which, when kept constant make it a deed. These are the object identifier of the creator, the date of creation, the content of the created document (so far it is analogous to paper certificates) and the dynamics of the process - another document, which plays the role of code in object oriented computer science.

Up to here the set of places in Askemos looks like a daunting disorder. Computer science tradition would try to press them into some hierarchy and there might be a good reason to do so from the cognitive point of view. People have a tendency to organise things in hierarchies and call it "making order".

At the other hand, global namespaces have a tendency not to work at all and this is not only due to different lexical ordering rules and preferred character sets. The author has spent countless hours in meetings concerning the document structure of large projects and found that personal preferences are the major hindrance to any settlement. The impact and tangle about the distributed name system in the Internet shall just serve as an example and the public interest in peer to peer systems, which maps naturally to human interaction, shall serve as another hint.

But from any users point of view, there is usually nothing wrong, if the namespace has it's root exactly in this user. After all this is exactly the mirror of the concept of 'I'.

In Askemos there is a set of names associated with every place from which named objects (ideas, concepts) are looked up. This association map serves a second purpose: a place nowhere referenced any more is forgotten and garbage-collected from the system after some time. (With respect to the environment, raw material from collected garbage is separated and recycled, and special care is taken of hazardous waste.)

In implementations only peer to peer mechanisms are to be used, nothing shall rely on global namespace or any other limited resource, which is inevitable for useful operation. Limited / global resources are, instead, understood as exported local resources, which clearly are subject to trade.

To summarize: the user space of Askemos consists of autonomous cells called "places". These cells have the ability to memorize values as their internal state and compute - in an atomic transaction - a new state and set of values to be send as messages to other places.

## Distributed Authority

One important aspect of the Askemos machine is a global rights, roles, and personal identity management system.

It has already been mentioned that a central authority is a major security risk due to threats posed by computer criminals and merely a systematic error of nowadays computer infrastructure. Furthermore, and this is of conceivable importance for application in the legislative domain, central authority is a mismatch with todays decentralized, democratic societies.

Capability based schemes have been proposed to reduce the dangers inherent to environments finally controlled by a superuser. In a capability system the authority of a program is just the capabilities it holds. A capability is usually an opaque bit pattern, which, when owned by an object, enables that object to invoke other objects, which in turn decide by themselfs whether they execute the requested operation. Given that the invoking object knows which operation it invokes, this is equivalent to the invoking object holding a capability for each operation it can invoke.

Processes must be confined to a strict protocol when manipulating capabilities. Known capability-based systems use a special trusted instance, the system kernel as in KeyOS, the object reference like the Java virtual machine or even some hardware support as IBM's System 38, to create a universe of capabilities and confine application code regarding capability operations. An open, distributed system lacks such a special status. In this section a rule set is derived which defines the capability manipulation rules for the Askemos system.

This protection scheme is an extension of capability systems. Instead of the usual opaque bit pattern, a rule is given, how to create new capabilities and grant those to, and revoked them from, other users. Furthermore, these capabilities are not constrained to invoke methods of certain objects, instead they can be used to protect any kind of method in any object. One consequence is that, while usually capabilities are also used to locate the receiver of messages, such an implicit knowledge is not encoded here. To distinguish this kind of extended capability from an opaque, unstructured value, we call such structured capability a "right".

Askemos uses a separate value space (special slots of messages and places) to hold rights. This is a rather arbitrary decision for the sake of clarity about the sensitive topic, especially in order to make it easy to discern rights from less sensitive data. Capability bits are - in contrast to some other systems

- easily read from their data space. Their manipulation is controlled via system calls.

Within Askemos messages are understood as orders. A set of rights is associated with a message, which indicates the authority under which the ordered operation shall be performed. Whenever a place responds to a message, the code may check whether those rights are sufficient and deny or execute the operation accordingly.

It should be noted that the scheme can not protect against dirty tricks, where users persuade others to unwittingly grant their rights.

**The Rules**

**Rights Managment**

> "Set theory can be viewed as a form of exact theology." [Rudy Rucker]

It is a bit more complicated to move rights around without a superuser, but not much. The best might be to imagine every single user as the superuser of itself. This leads to some simple rules: There is a set $A$ which contains all users $S$.

$$S \subset A$$

Every users set $S$ contains (among other ressources pertaining to that user) the set of this users own rights $r$.

$$r \subset S$$

Every atomic right $t$ is an element of $r$

$$t \in r$$

(Atomic rights are somehow similar to filesystems read, write, execute attributes.)

The set of rights required for an operation is $r_{needed}$.

$$r_{needed} \subseteq r$$

Every user $S$ can copy a subset $r_{given}$ of his own rights $r$ to another user $S'$ rights $r'$. After this operation we would have:

$$\forall\, t \in r_{given}\,(\,t \,|\, t \in r \wedge t \in r')$$

To avoid the creation of a superuser, a user must not copy all of its rights, especially not to one user alone. In fact, the user must not even be able to do so accidentally. This leads to the condition that the set of all copied rights $r_{copied}$ must not equal the set of all of the users rights $r$.

$$\neg(r \backslash r_{copied}) = \emptyset$$

It is now easy to check the rights for a certain document by testing for equality:

$$(r \cap r') = r_{needed}$$

This operation is too expensive in an implementation, so the set $r_{given}$ remains intact, as a set in $r'$. We can do then a more convenient check:

$$(r_{given} \cap r_{needed}) = r_{given} = r_{needed}$$

These conditions apply – equally well – to the set of rights $r_{given}$. There exists a superset of $r_{given}$, called $r_{owned}$, which stands for the right to delete $r_{given}$. User $S$ withholds the difference of $r_{owned}$ and $r_{given}$.

$$\forall\, r_{given}\, \exists\, r_{owned}\,(r_{owned} \,|\, r_{owned} \supset r_{given})$$

With the definitions given so far, the necessary parts of the rights system are complete. Each user is the owner of an extensible realm of rights $r$. Individuals can grant and revoke parts of their realm among each other. And none can aquire all the rights owned by anybody else.

**Rationale and Background**

The actual system as implemented, is - for the sake of both simplicity of implementation and performace - even more restrictive than described so far. On the other hand it provides some more functions to simplify testing for common cases. Future versions might relax the rules, up to the limits given in the lst section.

The first idea for this protection scheme was derived from the one implemented in VSTa and worked very well with an earlier prototype.

There is, however, a chicken and egg problem: How are new users (and their initial rights) introduced to the system?

Users cause processing and need resources of those physical machines, at which agents support their places. The physical machine is a property of it's owner and, hence, the owner should have the final authority to allow other people to utilize his/her ressources under negotiable terms. Therefore the owner has to have the chance to allow and terminate support for other users (to be regulated by real world contracts).

This is done by allowing the owner to link user places to the one place, which represents the particular machine.

### No single Point of Failure, Byzantine Protocols in a P2P world

In a distributed system it is meaningless to assume all parties to be honest and reliable. Consequently, there must be no need to ever trust any single machine completely for the virtual machine to continuously operate correctly.

In the absence of byzantine protocols every computer models the application. Any failure of this machine and any malicious manipulation will inevitably effect the application. Byzantine agreement, has been identified as one of the core building blocks for the design of reliable distributed systems. Here the model is a "local opinion" only, a projection of a non-physical process to a physical model. In the case of concordance of a majority of parties about these models, it can be assumed that they are correct.

Several proposals and papers successfully demonstrated the utility of byzantine protocols. Most, if not all of them, rely on asynchronous messages only, which was another reason to favor those in the definition of the virtual machine. Please refer to the study "Secure Intrusion-tolerant Replication on the Internet" [4] by C. Cachin and J. Poritz for more details about byzantine protocols.

An important problem with distributed settings is the grouping of parties into coalitions for the byzantine aggrement process. One point to consider is that the selected parties inevitably have access to the clear text of the informational process. Until it is known, how processing in the absence of that knowledge can be performed, the owner of the information must trust the supporting parties. Such a situation translates into a shared secret, which has to be distributed among the selected parties, or, alternatively the unification of several secrets at an upper level.

The threat of physical take over, for which no amount of cryptography can ever compensate, requires the final authority about such party selection to be left with the owner of the information. There is nothing wrong with that. After all, safe-keeping is a business in the real world, why should that be different in the virtual world?

Any machine participating in an Askemos network will always be owned somehow, be it the registry office archive of a town or the calendar and diary in a mobile phone. Fees or tax will have to be payed for the former and personal trust and support will count for the spouses phones to have each others calendars - or just not. A wide range of motivations on human level justify the decision, how exactly to distribute information. Hence the question is out of the realm of computer science.

One more technical motivation to assign the act of trust distribution to a human stems from the fact that it is much cheaper to account for non-cooperative byzantine failures in the protocol than for cooperative failures. Cooperative failure translates into simultaneous malicious operations of different parties. This is a situation, which a human assignment can easily judge about from physical locations and ownership reputation, while hardly expressible "in" computers.

## Reality

> "What is the difference between theory and reality? - In theory there is none."

Reality requires further decisions. This section gives an overview of the current prototype implementation. It should be noted that many details presented here could be subject to discussion, while the overall semantics stay intact.

All requirements and aspects described so far translate directly into some code. Decisions how exactly to implement the aspects were done by selecting the best practices from standard technology. When calling them best, this means here, that they are sound in theory and, if that criterion leaves more than one way, all should be possible, but the most successful one was chosen. Furthermore, care has been taken to introduce as few dependencies as possible, while still providing a practically useful environment.

The aim behind this decision strategy is to avoid creating technical, economical or social deployment hindrances, without sacrifying the theory behind.

### Host System

The existing Askemos system does not yet control the hardware directly. It is instead implemented as a server, which at least runs at POSIX compatible systems, even though most of their functionality is not needed at all. Therefore, it can be easily deployed on more secure servers of the Internet, as well as on cheap hardware running a GNU/Linux or BSD based system.

The caveat: those host systems still have a superuser. For a high quality deployment, no further services should be run on the Askemos server machine and a superuser login or the execution of arbitrary commands via other mechanism like the `su` command should be disabled.

Size and complexity of the whole system indicates that it's feasible today to port Askemos to cell phones, handhelds and similar hard ware.

### Programming Languages

As mentioned before it's feasible to provide any language for application programming. It just has to be done in such a way that the program is confined to an isolated environment, which can be a burden with some language implementations.

The prototyping language Scheme, which is currently also used for implementation, was chosen because it is a simple and small language, with provisions for safe computing (range checks, no pointers) and capable abstraction mechanisms (higher order functions, continuations). It is well standardized and the standardization process proceeds. Many implementations exist and are freely available for all kinds of environments, for instance down to organizers hardware, the Java virtual machine and as new as the .NET platform. Some compilers are freely available, which create efficient C code and interfacing to legacy C code is usually not hard. To summarize: it's cheap and always possible to migrate to another environment.

Furthermore Scheme is a direct ancestor of the DSSSL (ISO 10179:1996) standard, which in turn has influenced the XSLT recommendation of W3C. The latter are both pure functional languages and thus a good fit for the Askemos virtual machine. Scheme as the ancestor made it easy to provide those languages at application level.

The danger of making mistakes is largely reduced by implementing the Askemos kernel almost completely in functional style as well. This makes it also easy to guarantee that places are confined to their data space. There is no need of address spaces from the host system, as a means to that end.

The selected implementation, rscheme, provides also preemptive threads (essential for the asynchronous execution model).

### Data Model and Compatibility

The Askemos virtual machine is neither data model specific, nor is the implementation. Currently, the market makes strong movements towards settling with XML as the universal data exchange format. XML is capable to express all kind of tree structured data, well standardized and widely deployed. Therefore XML was selected as the "native" data format of the implementation; processing XML is optimized.

Settling with XML removes also most compatibility barriers, as more and more legacy applications provide XML encoding for data exchange.

The data is currently held in two storage medias: a) A persistent store, which keeps the tree structures, hash tables etc. in the internal form as accessible by the executable code. The technique is called "pointer swizzling at page fault time". And b) in a file system representation.

### Network Protocols

The most important application protocols on the Internet are HTTP and SMTP. The current implementation relies on just these protocols, thereby interfacing to most applications which are used for human communication.

Because of the importance of these protocols, most firewalls somehow allow such traffic, so no obstacles are introduced. Though utilizing SMTP for voting in byzantine protocols would probably go to the effect to grind the system to halt.

All place to place communication is conceptually understood as SOAP messages. It is called conceptually here, because several redundant information (envelope and headers) is not actually generated until it's really needed.

### Legal Long Term Availability

All data formats, programming languages and other standards used for Askemos are unencumbered by any claims of intellectual property, which is not freely available. No sudden changes in anybody's mind can impact the utility of the system to others. The implementation itself and the underlying language is available under the gnu public license.

## Privacy

In the course of the past years some effort was made to establish global user identity and data repositories. Those appears to be a premise for electronic trade systems and would ease many applicative tasks. Even though no final judgement can yet be made about their success, it appears that they face heavy resistance from potential customers and users up to a degree that those projects are put to a rest.

Said resistance apparently stems from the human interest in privacy. The account aggregation those proposed schemes incur do not only drive paranoid security experts away, but are also felt by the casual users. Askemos, being distributed without super user authority, can achieve the same utility regarding trade while preserving safety, security and privacy. It just has to be completed with a decent contract system, to assure quality of service. The users are than free to choose whom they trust with their information.

## Conclusion

The Askemos system has been practically proven as a deployable operating system. It's components are all well researched, none relies on unsound or unproven premises and they don't interfere. The system definitions utilizes only public standards. Finally all components are available and safe from surprising intellectual property claims, a premise to avoid a digital divide.

## References


[1] J. F. Wittenberger, "The Askemos project" 1999-2002.
`http://www.askemos.org`

[2] I. Clarke, "The freenet project" 1999-2001.
`http://freenet.sourceforge.net/`

[3] J. Kubiatowicz et. al, "The ocean store project" 2000-2001.
`http://oceanstore.cs.berkeley.edu/publications/`

[4] C. Cachin, J. A. Poritz, "Secure Intrusion-tollerant Replication on the Internet" 2002.
`http://www.zurich.ibm.com/čca/papers/sintra.ps`

[5] Donovan Kolbly, `http://www.rscheme.org`

[6] V. Singhal, S. V. Kakkad, P. R. Wilson, "Texas: An Efficient, Portable Persistant Store"
`ftp://ftp.cs.utexas.edu/pub/garbage/texaspstore.ps`

[7] H. Abelson and G. J. S. with Julie Sussmann, "Structure and Interpretation of Computer Programs" MIT Press, 1985.

[8] `http://www.daimi.au.dk/PetriNets`

[9] The Free Software Foundation "GNU Public License"
`http://www.gnu.org/licenses/gpl.html`