# Askemos - eine verteilte Umgebung

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

28 April 2002

**Zusammenfassung**

Dieser Artikel präsentiert Askemos[1], ein autonomes, verteiltes Betriebssystem auf Basis von Peer-to-Peer Netzwerken[1]. Askemos funktioniert im Vergleich zu derzeit marktüblichen Betriebssystemen auf signifikant höherer Abstraktionsebene. Askemos bietet eine "enabling technology" für "e-Integration" von Geschäfts-, Abrechnungs- und Workflowprozessen zwischen Unternehmen. Eine flexible und sichere Infrastruktur zur fälschungssicheren Verarbeitung von Informationen unter Beachtung intellektueller Eigentumsrechte.

Askemos definiert eine einfache, abstrakte Architektur: eine virtuelle Maschine auf Dokumentenebene. Diese virtuelle Maschine hat keine Repräsentation auf einer einzelnen physischen Maschine. Vielmehr operiert diese Maschine in der "Vorstellung" einer Menge von kollaborierenden Agenten, welche die virtuelle Maschine zu beobachten scheinen. Um diesen Effekt zu erreichen, berechnen die Agenten die Einzeloperationen der virtuellen Maschine unabhängig voneinander (gleichberechtigt) und stimmen den als wahr verstandenen Zustand untereinander ab.

Illegalen Angriffen und repressiver Ausbeutung individueller Ressourcen wird vorgebeugt, indem keine unikaten Ressourcen wie zum Beispiel zentrale Autoritäten, Administrationen existieren.

## Einführung

Das digitale Zeitalter startete mit verschiedenen Versprechen. Neben einer Menge kurioser und abstrakter Träume gab es ernstere Pläne. Einer davon war immer das "papierlose Büro". Eine humorvolle Definition dazu sagt heute, es sei dasjenige mit dem höchsten Rabatt bei der Papierhandlung. Solche Witze enthalten stets ein Körnchen Wahrheit.

Seit der Erfindung der Schrift werden Medien benötigt, um Informationen zu speichern. In dem Maße, in dem das Informationsvolumen anwuchs, wurden Masse und Kosten des Mediums von Stein über Ton, Papyrus und Pergament bis zum heutigen Papier reduziert.

Es ist nur logisch, die Möglichkeiten der Computertechnik entsprechend anzuwenden: Computer können nicht nur soviele Informationen wie das Papieräquivalent eines mittelgroßen Waldgebietes speichern sondern diese Informationen darüber hinaus auch noch bearbeiten.

Jedoch wird Papier immer noch bevorzugt! Da stellt sich die Frage, was an der heutigen Computertechnik als für diese Zwecke unzureichend erlebt wird. - Eine kurze Analyse zeigt: sie ist nicht zuverlässig genug.

Selbst ein säuregebleichtes Stück Papier ist nach 20 Jahren noch lesbar. Versucht man dies mit digital gespeicherter Information, so stellt man fest, daß - ganz abgesehen von physischer Beschädigung der Datenträger - die gute Chance besteht, daß das Datenformat praktisch nicht mehr lesbar ist. Dabei handelt es sich lediglich um eine einfache, textuelle Definition, die technisch nicht mehr verfügbar ist!

Beispiele gibt es genug: Software entwickelt sich und Datenformate werden geändert. Alte Software hört aus irgendwelchen Gründen auf zu arbeiten - spätestens wenn sie aktualisiert wird. Wenn dann die neue Version alte Daten nicht mehr vollständig erkennt und niemand beauftragt werden kann, das Problem zu beheben, dann sind die Daten verloren. Mit etwas Glück und wenn die Word-Prozessoren von der selben Firma stammen, dann kann man fünf Jahre alte Dokumente in neuere Word-Prozessoren importieren. Allerdings wird man wahrscheinlich einen guten Teil der Formatierung verlieren. Im ungünstigsten Fall ist gar nichts mehr zu lesen und alle Daten sind verloren.

Das ist natürlich mit 20 Jahre alten Daten noch viel wahrscheinlicher.

Oder man stelle sich einen Vertrag vor, der schon lange Jahre auf der Festplatte eines Computers liegt.

[1] Peer-to-Peer Netzwerke sind solche, in welchen keine Komponente eine zentrale, unersetzbare Rolle einnimmt.

Eines Tages sagt der Sprößling: “Schau, welch’ cooles neues Spiel! Übrigens mußte ich etwas Platz schaffen, doch keine Sorge, ich habe nur gaaanz alte Dateien gelöscht...” Sollte das nicht vermeidbar sein?

Zum Vergleich dazu ein Blick in die heutige Welt; Dokumente wie Ausweise und Geburtsurkunden werden zu Papier gebracht. Um einen Ausweis ausgestellt zu bekommen, braucht man eine Geburtsurkunde. Die Geburtsurkunde gehört Ihnen, dem Eigentümer. Es verbleibt jedoch eine Kopie auf dem Standesamt. Geht die eigene Kopie verloren, so kann Ersatz besorgt werden. So bleibt eine Kopie für den Ausweis zurück und sicher weitere zu anderen Zwecken. Dies sichert die Information gegen Fälschung und Zerstörung des Mediums. Eine Kopie ist gar nichts; die selbe Information auf mehreren Kopien aus mehreren Quellen ist hingegen ein Beweis. Es gibt keine 100%ige Garantie, daß es nicht doch eine Fälschung ist. Jedoch ist die allgemeine Auffassung, daß es sehr unwahrscheinlich ist, daß jemand wichtige Informationen fälschen oder rnichten kann.

Ein weiterer Schluß ist, daß eine einzelne Kopie keinen Wert besitzt. Man versuche eine Geburtsurkunde zu fälschen und im Standesamt zu bitten, deren Kopie zu aktualisieren. Der Beamte mag dazu im Stande gesetzt werden können, durch Bestechung oder gar Erpressung, jedoch dürfte es schwer werden, alle Spuren des Betruges zu beseitigen.

Auch sei erinnert an all die Nachrichten über “gehackte” Web-Sites sowie gestohlene Daten und es sollte offensichtlich werden, warum die Menschen den Computern bezüglich der Korrektheit sensibler Daten nicht trauen.

Das heißt nun nicht, daß Computer wirklich nutzlos sind. Als die Bibliothek in Alexandria niederbrannte, ging der größte Teil des Wissens verloren, da nur wenige Kopien der Werke anderswo aufbewahrt wurden. Hier können Computer wirklich helfen, indem Daten in verteilen Speichersystemen wie freenet[2] oder ocean store [3] gehalten werden.

Askemos ist ein Versuch, ein Spezialbetriebssystem zu definieren, maschinenunabhängig und verteilt. Der Schwerpunkt liegt dabei auf der Vertrauenswürdigkeit. Es soll für jene Anwendungsfälle passen, in denen die heutige Computer-Infrastruktur Unzulänglichkeiten offenbart: amtliche Aufgaben der Verwaltung, Redefreiheit, Langzeitverfügbarkeit von Wissen, Handel und Zahlungsverkehr, Gerichtsbarkeit u.ä. Diese Anwendungsgebiete haben einen gemeinsamen Punkt: Informationen sollen - selbst im Fall physischer Beschädigung - sicher und nachprüfbar korrekt sein und zwar so weit, daß die Menschen ihnen letztlich Beweiskraft zuerkennen. Um dies zu erreichen, wurde versucht nichts absolut Neues zu erfinden, vielmehr wurden die Regeln des “Papierzeitalters” auf die digitale Welt übertragen.

Askemos muß sicher und nicht korrumpierbar sein. Der Autor fühlt sich persönlich herausgefordert einen Weg zu finden, um das System zu brechen. Derzeit hat er jedoch keine Idee, wie das möglich sein soll. Damit die Sicherheit des Systems nachprüfbar ist und eventuelle Fehler gefunden werden können steht der Quellcode des Systems steht öffentlich zur Verfügung.

> “Security through obscurity does not work!”

## Gefahren und Anforderungen

Informationelle Ressourcen und Prozesse müssen verschiedenen Gefährdungen wiederstehen um Zuverlässigkeit und Vertrauenswürdigkeit (Beweiskraft) zu erzielen. Erforderlich sind:

**Sicherheit (Safety) und Verfügbarkeit** Informationen können durch massives Anlegen von Sicherungskopien bzw. in verteilten Speichersystemen wie freenet zuverlässig verfügbar gehalten werden. Um illegale und versehentliche Manipulationen zu verhindern, müssen jedoch beim Ändern strenge Regeln durchgesetzt werden.

**Manipulation und Verlust muß um jeden Preis verhindert werden.**

**Sicherheit (Security)** Proprietäre und private Informationen müssen gesichertes und geheimes Gut im Eigentum sein. Diese Anforderung verbietet schon die schiere Existenz jedwiger administrativer Macht, welche auf technischem Wege persönliche Entscheidungen des Eigentümers nivellieren kann.

**Verbindlichkeit** Zusammen mit der Forderung nach einer Privatsphäre kommt die rein rechtliche Forderung, daß die Verwendung illegal erhaltener Informationen ein illegaler Akt ist. Um diese Regel effektiv durchsetzen zu können, ist ein gewisser Grad an Verbindlichkeit und entsprechende Nachverfolgbarkeit erforderlich.

Weiter benötigen Handel, Zahlungsverkehr und amtliche Anwendungen zwingend die Verbindlichkeit entsprechender Prozesse.

**Integrität** Eine allgemeine Grundnotwendigkeit, die gewöhnlich erfüllt wird. Integrität meint, daß keine überraschenden Änderungen an Daten erfolgen können. Entsprechende Tests sichern üblicherweise gegen Fehler bei der

Übertragung und Speicherung. In Askemos werden diese weiterhin eingesetzt, um neben der Integrität der reinen Daten sicher zu stellen, daß verbindlichkeitsrelevante Metadaten unverfälscht bleiben.

Dieser Artikel beginnt mit dem Konzept der virtuellen Maschine von Askemos, danach werden einige ihrer Details näher erklärt. Letztlich folgen Implementationsentscheidungen und Bemerkungen zum Entwicklungsstand des aktuellen Prototyp.

## Zuverlässigkeit im Fadenkreuz

Der Askemos Kern liefert Denotationssysteme (minimale Sprachen) für die drei "Achsen" von Information: Daten (sichtbare, interne Struktur), assoziativer Kontext (black box Umgebung) und Rechte. Die wichtigste Systeminvariante ist dabei Vertrauen.

Der Askemos Kern ist eine Kommunikationsinfrastruktur. Als solche muß er unausweichlicher Weise im Stande sein, alle Arten von Kommunikationsakten zu unterstützen, was wiederum impliziert, daß er auch für unpopuläre und kriminelle Anwendungen geeignet ist - eine inhärente und unausweichliche Eigenschaft jeder Infrastruktur. Die Notwendigkeit universell verfügbar zu sein und der Fakt, daß Sicherheit durch Verschleiern nicht funktioniert, waren die primären Gründe den Quellcode unter der GNU Public License[9] verfügbar zu machen.

Es folgen Anforderungen, die für den Kern definiert wurden:

- Vertrauenswürdigkeit ist das ultimative Designziel. Alle Komponenten sind mit dem Kriterium der Nachprüfbarkeit ausgewählt. Sollte dieses Kriterium mit praktischen Wünschen kollidieren, so erhält es Vorzug. Integrität wird durch adäquaten Einsatz kryptographischer Algorithmen und Protokolle sichergestellt. Es dürfen keine Annahmen über die Ehrlichkeit der Eigentümer der zugrundeliegenden Hardware oder der beteiligten Nutzer notwendig sein. Auch wird keinerlei Sicherheit der Hardware im Netzwerk vorausgesetzt, da eine solche Voraussetzung unrealistisch ist.

- Das System erreicht Verfügbarkeit durch redundante Verteilung in dem Maße, wie es der Anwnedungsfall verlangt.

- Ein Schutz- und Rechteschema, welches bewiesenermaßen nicht brechbar ist, wird eingesetzt, um private Sphären der Information zu schützen. Eine Nachträgliche Analyse von Maschinen aus dem System darf nicht mehr Informationen preisgeben als exakt jene, die von den Agenten dieser Maschinen unterstützt wurden.

- Kein einzelner Schadensfall darf im Stande sein, das gesamte System in der Funktion zu behindern.

- Alle Prozesse und Objekte sind "im System", es darf keine Ausnahmen geben. Objekte sollen vollständig und minimal (small is beautiful), m.a.W. im strengen Sinne korrekt abstrahiert sein.

- Das System ist in technologieunabhängiger Weise unter Anwendung weit verbreiteter und offener Standards implementiert.

Sollte eine Implementation diesen Anforderungen nicht genügen, dann ist dies ein Fehler, egal wie die Abweichung begründet ist.

Anforderungen an Anwendungen auf der Nutzerebene:

- intuitiv, einfach und verständlich

- enge Koppelung zwischen Entwicklern und Nutzern (Synergieeffekt).

- Modellierte Dokumente sollen sich "papierartig" verhalten.

- direkte Manipulation von Objekten soll möglich sein, insbesondere der Handel mit diesen

- Alternativen und Migrationspfade sind Teil jedes Designs.

Es ist anzumerken, daß diese Anforderungen eine Intention darstellen. Als der Artikel geschrieben wurde, existierten erst wenige Anwendungen.

### The Distributed Virtual Machine

Das Schlüsselkonzept von Askemos ist die verteilte, virtuelle Maschine, welche informationelle Prozesse abwickelt. Diese Maschine ist abstrakt definiert in Begriffen von Dokumenten und Metadatenverwaltung.

Die verteilte virtuelle Maschine von Askemos funktioniert im Informationsraum, einem abstraktem virtuellem Raum in dem Information gedacht wird, unabhängig von Medien und physischem Aufenthalt. Zur Erläuterung des Konzeptes stelle man sich ein Fernschachspiel vor. Zwei Schachspieler und ein Schiedsrichter in einer Telefonkonferenz. Um den Punkt noch mehr herauszustellen, wird von

allen Beteiligten verlangt, sich das Spielbrett lediglich vorzustellen. Hier findet nur ein einziger informationeller Prozeß statt: Wir verstehen den Vorgang als ein einziges Schachspiel, nicht drei verschiedene. Die elektrischen und chemischen Signale in den Gehirnen und Telefonen werden als Projektionen des Prozesses in ein bestimmtes Medium verstanden. Nimmt man nun ein physisches Schachbrett und ein Transkript der Partie auf Papier dazu, so sind dies lediglich weitere Projektionen der selben Information in einem jeweils anderen Datenraum.

Das selbe Prinzip findet in der virtuellen Maschine von Askemos Anwendung. Die Maschine existiert im Informationsraum und wickelt dort ihre Prozesse ab. Auf physischer Ebene partizipieren verschiedene sogenannte Agenten. Diese berechnen ihre individuelle Vorstellung vom Zustand der Maschine und stimmen diese untereinander ab. Diese "individuellen Vorstellungen" werden als Projektion derselben abstrakten Information verstanden, egal wie unterschiedlich sie kodiert sein mögen. Beispielsweise kann ein Agent seine Projektion eines Bildes als PNG-Datei bewahren, ein anderer im JPEG-Format. Es ist immer noch das selbe Bild, wie menschliche Intelligenz feststellen kann, ohne das Bild in einer der beiden Formen zu analysieren.

Wenn nun ein Agent, sei es eine physische Maschine oder ein Mensch, einen anderen Zustand der virtuellen Maschine annimmt, als rational als korrekt bewiesen werden kann, so ist dieser Agent im Irrtum. Die Projektion (Glaube) dieses Agenten wird als falsch angesehen - zumindest vom Standpunkt des Beobachters. Was passiert nun, wenn sich das Wissen der Menschheit über den in Frage stehenden Prozeß weiter entwickelt und eine neue formale Spezifikation des Prozesses gegeben wird? Dann läuft der alte Prozeß immer noch weiter, denn er ist nach der alten Auffassung immer noch korrekt. Die neue Auffassung ist jedoch bereits im Informationsraum und Askemos' virtuelle Maschine führt beide, die alte und die neue Version, gleichzeitig aus. Auch wenn die alte Version in der Praxis selten verwendet werden wird.

Askemos' virtuelle Maschine kann die zuvor angegebenen Anforderungen (Sicherheit, Zuverlässigkeit, Verbindlichkeit und Integrität) erfüllen, denn diese können formal spezifiziert werden. Die Verfügbarkeit der Information ist aber immer noch gefährdet, falls zu wenige Projektionen (Kopien) zur Synchronisation bereit stehen. Dies ist jedoch eine statistische Frage und kann auf die erforderlichen Ausfallwahrscheinlichkeiten eingestellt werden.

Die folgenden Abschnitte präsentieren die Definition eines minimalen (und hoffentlich vollständigen) Satzes von Aspekten (hier auch "Achsen") der virtuellen Maschine. Es wurde darauf geachtet, daß eine formelle Spezifikation und Validierung auf der Basis bekannter Theorien möglich ist.

## Plätze im Informationsraum

Es wurden bereits viele Taxonomien vorgeschlagen, um jene Begriffe zu strukturieren, welche benutzt werden, um Ideen auszudrücken. Dies gilt insbesondere im Bereich der Programmierung. Läßt man letztere beiseite (um den Wald vor lauter Bäumen überhaupt noch sehen zu können), so kann man festellen: Eine Idee ist eine Menge von Assoziationen von Eigenschaften zu Werten. Diese Assoziationen werden gewöhnlich Paare, Bögen (arc) oder Verweise (arrows) genannt. Wenn solch eine Menge eine Idee repräsentiert, so kann sie identifiziert werden, besteht dauerhaft und wird unter gewissen Umständen wieder vergessen. In den Begriffen der "computer science" ist dieses Konzept als "Frame" bekannt. Dies ist die Basiseinheit von Askemos. Ein gut bekanntes Konzept, auf das deswegen hier nicht weiter eingegangen wird.

Mit Blick auf die menschliche Erfahrungswelt und die natürliche Präferenz für dreidimensionale Modelle, also im Interesse der menschlichen Benutzer, nennen wir diese Basiseinheit in Askemos einen "Platz". Die Askemos Maschine garantiert (eine minimale Menge von) Invarianten in den Eigenschaften eines jeden Platzes.

Plätze haben die Eigenschaften von Prozessen, Objekten bzw. Agenten in anderen Betriebssystemen. Sie empfangen Nachrichten, reagieren auf diese mit Änderung ihrer Eigenschaften (gesteuert von der virtuellen Maschine) und senden Nachrichten an andere Plätze aus. Dies erfolgt in einer atomaren Operation: dem gut bekannten Prozeßschritt. Plätze sind autonom: dies ist der einzige Weg, um die Eigenschaften eines Platzes zu ändern.

Eine Nachricht hat die gleiche Frame-Struktur wie ein Platz. Allerdings sind Nachrichten nicht persistent und können auch nicht direkt identifiziert werden, sie sind entweder vorhanden oder nicht.

Letztere Entscheidung mag hier etwas arbiträr aussehen, wenngleich sie die Semantik von Nachrichten mit der Programmiersprache Erlang, dem SOAP-standard und ähnlichen Modellen teilt.

Abgesehen von den weiter unten angeführten Argumenten hält der Autor RPC für ein fehlerhaftes Design. Es gibt zahllose Kommunikationsprozesse und Medien in der realen Welt: Nervensysteme, Signalstoffe wie sie von Ameisen und Bienen verwendet werden, elektrische Schaltkreise, menschliche Stimme und Briefe. Alle sind unidirektional und asynchron - der Briefträger wartet nicht auf die Antwort zu jedem Brief den er ausliefert. Mit asynchronen, unidirektionalen Nachrichten ist ein breiter Bereich an Reaktionen auf beiden Seiten des Informationskanales möglich. Beide Prozesse können im Feh-

lerfall weiterlaufen. RPC wurde gezielt entwickelt, um die Verteilung von Objekten zu verbergen und Prozeduraufrufe so zu präsentieren, als wären sie lokal. Dieses Absicht wird gut erreicht, aber es ist das falsche Ziel, denn letztlich folgt daraus eine arbiträre Einschränkung.

Erstens kann dieses Nachrichtenmodell einfach auf die Elemente der Petrinetz Theorie[8] abgebildet werden, welche recht mächtig ist um die Dynamik des Modells zu beschreiben. Dadurch kann die Dynamik des Nachrichtenaustausches zwischen Plätzen formal beschrieben und analysiert werden. Bestimmte wichtige Eigenschaften wie Freiheit von Dead- und Livelocks können abgeleitet werden. Dazu nehmen die Daten des Platzes die Rolle der Plätze der Petrinetz-Theorie an, Zustandsänderungen werden durch Transitionen dargestellt und der Inhalt der Nachrichten entspricht den Petrinetz-Marken.

Zweitens berechnet das Modell das Ergebnis einer Transaktion in einer atomaren Operation von nur zwei Parametern: dem aktuellen Zustand der Eigenschaften eines Platzes und den Eigenschaften der eingehenden Nachricht. Gleichgültig welche Programmiersprache verwendet wird um die Berechnung selbst zu programmieren, bildet dieser Effekt die Auswertung einer Funktion. Nach Wissen des Autors ist rein funktionale Programmierung, also ohne Seiteneffekte, die einzige Möglichkeit, formale Beweise der Korrektheit von Programmen zu ermöglichen[7] Seite 175. Dies ist damit hier gegeben.

Drittens teilt das gewählte Modell Prozesse in Prozeßschritte ein, welche für die unterliegende Maschine sichtbar sind. Werden diese mit vernünftiger Granularität (Zeitaufwand zur Berechnung eines einzelnen Schritts) gewählt, so erhält man eine günstige rücksetzbare Operation zur Synchronisierung der verschiedenen Projektionen (Kopien) eines Platzes (siehe auch Abschnitt über byzantinische Protokolle).

Dieser Moment der Synchronisation ist weiterhin sinnvoll, um den Prozesszustand in verschiedenen persistenten Speichern zu hinterlegen. Es ist ein bekannter Fakt, daß die Zeit zum Speichern mit der Zuverlässigkeit des Speichermediums steigt. Der Hauptspeicher verfolgt jede Änderung des Prozessors - unmöglich mit der Festplatte, ganz abgesehen von verteilten Speichermedien wie freenet. Im Moment der Synchronisation wird daher ein Schnappschuß genommen. Während der Prozeß schon fortschreitet, wird danach der Zustand in den persistenten Speicher geschrieben.

Vergleicht man schlußendlich das Modell mit neueren Arbeiten über “intrusion tolerant replication system” [4], so wird deutlich, daß die benötigten primitiven Operationen (wie beabsichtigt) exakt mit jenen korrespondieren, die von byzantinischen Replikationsarchitekturen bereitgestellt werden.

## Topologie und Assoziationskontext

Jeder Platz hat einen global eindeutigen Identifikator. Dieser sollte, um Verbindlichkeit sicherzustellen, in irgendeiner Form eine Funktion des Inhaltes und der Metadaten des Platzes sein. Dadurch wird Fälschung unmöglich bzw. mindestens feststell- und nachverfolgbar, da Dokumente ihren Platz verlassen, d. h. ihren Identifikator ändern, sobald der Inhalt modifiziert wird. Ein solches Verhalten ist jedoch für Prozesse nicht geeignet, da sie für praktische Belange unadressierbar werden. Aus diesem Grund wurde entschieden, oben genannte Invariante nur für authentische Dokumente, also solche, die konstante, unfälschbare Fakten enthalten, zu erhalten. In der Praxis handelt es sich bei diesem Identifikator um eine kryptographische Prüfsumme von jenen Eigenschaften des Platzes, welche urkundenrelevant sind, sofern sie konstant bleiben. Konkret sind dies der Identifikator des Urhebers, der Zeitpunkt der Erzeugung und der Inhalt (soweit das identisch ist mit papierbasierten Urkunden) sowie das dynamische Verhalten des Prozesses - ein weiteres Dokument, welche der Rolle des Programmcodes in objektorientierter Modellierung entspricht.

Bis hierher scheint die Menge von Plätzen im Askemos eine furchterregende Unordnung abzugeben. Herkömmlicher Weise würde man versuchen, diese in eine, wie auch immer geartete, Hierarchie zu pressen und es gibt, ausgehend von kognitiven Aspekten, guten Grund dies zu versuchen: Menschen versuchen Dinge hierarchisch anzuordnen und nennen dies “Ordnung machen”. Der Grund liegt vermutlich einfach in der hierarchischen Sozialstruktur des Herdentieres Homo Sapiens.

Andererseits haben globale Namensräume nicht nur wegen unterschiedlicher Ordnungssyteme für Schriftzeichen und bevorzugter Zeichensätze eine Tendenz, grundsätzlich nicht zu funktionieren. Der Autor hat viele Stunden in Meetings verbracht, in denen die Dokumentenstruktur größerer Projekte festgelegt werden sollte und dabei festgestellt, daß persönliche Präferenzen der Haupthinderungsgrund für jegliche Einigung waren. Der Wirrwar um das DNS des Internet und seine Auswirkungen seien hier als ein Beispiel zitiert. Auch kann das erhebliche Interesse der Öffentlichkeit an Peer-to-peer Systemen, welches in natürlicher Weise zu zwischenmenschlicher Interaktion paßt, als ein Hinweis verstanden werden, wie Information organisiert werden sollte.

Aus der Sicht eines jeden einzelnen Nutzers ist es geradezu perfekt, wenn der sichtbare Namensraum

seine Wurzel in genau diesem Benutzer hat. Letztlich entspricht das genau dem Konzept des "ich".

Im Askemos existiert zu jedem Platz eine Menge, in der benannte Objekte (Ideen, Konzepte) zusammengefaßt sind. Diese Menge von Assoziationen hat einen zweiten Zweck: ein Platz, der nirgends mehr referenziert wird, ist vergessen und wird vom System nach einiger Zeit aussortiert (garbage collection).

Alle Implementationen werden ausschließlich Peer to Peer Mechanismen verwenden. Keine für die normale notwendige Funktion wird von einem globalen Namensraum oder anderen beschränkten Ressourcen abhängig sein. Begrenzte bzw. globale Ressourcen werden statt dessen als exportierte lokale Ressourcen verstanden, welche eindeutig ein Handelsgegenstand sind.

Kurz zusammengefaßt besteht der Raum im Askemos aus autonomen Zellen, welche Plätze genannt werden. Diese Zellen können Werte als ihren internen Zustand speichern und berechnen - in einer atomaren Transaktion - einen neuen Zustand sowie eine Menge von Werten, welche als Nachrichten an andere Plätze gesendet werden.

## Verteilte Autorität

Ein bedeutender Aspekt der Askemos Maschine ist ein globales Verfahren zur Verwaltung von Rechten, Rollen und persönlicher Identität.

Wie bereits erwähnt, ist eine Zentralgewalt immer ein wesentliches Sicherheitsrisiko, da sie in krimineller Weise mißbraucht werden kann. In heutigen Computerinfrastrukturen ist dies dem Grunde nach ein systematischer Fehler des Systems. Weiterhin, und dies ist von offensichtlicher Bedeutung für Anwendungen im juristischen Bereich, paßt eine Zentralgewalt nicht zu den dezentralisierten, demokratischen Gesellschaften dieser Tage.

Es wurden capabilitybasierte Verfahren vorgeschlagen, um die Gefahren zu minimieren, die in Umgebungen existieren, welche letztlich durch einen Superbenutzer reguliert werden. In einem Capabilitysystem hat ein Programm lediglich die Autorität jener Capabilities, welche es besitzt. Eine Capability ist ein opaques Bitmuster, welches das besitzende Objekt berechtigt andere Objekte zu benutzen. Jene wiederum entscheiden selbst, ob sie die angeforderte Aktion ausführen. Da das verwendende Objekt die angeforderte Aktion kennt, ist die äquivalent dazu eine Capability für jede Operation zu besitzen, die verwendet werden kann.

Um Capabilities zu manipulieren, muß ein strenges Protokoll durchgesetzt werden. Bekannte capabilitybasierte Systeme verwenden eine besonders vertrauenswürdige Instanz, den Systemkern in KeyOS, die Objektreferenz in der Java virtuellen Maschine oder gar Hardwareunterstützung wie in IBM's System 38, um ein Universum der Capabilities zu modellieren und Anwendungsprogramme in ihren Möglichkeiten zur Manipulation zu beschränken. Einem offenen, verteilten System steht ein solcher spezieller Mechanismus nicht zur Verfügung. In diesem Abschnitt wird ein Regelsatz zur Manipulation von Capabilities im Askemos System abgeleitet.

Dieses Schutzschema ist eine Erweiterung der capabilitybasierten Systeme. Anstelle eines opaquen Bitmusters wird eine Regel angegeben, wie neue Capabilities erzeugt werden können sowie Regeln, wie diese an andere Benutzer vergeben bzw. diesen wieder entzogen werden können. Weiterhin sind diese Capabilities nicht darauf beschränkt, Methoden bestimmter Objekte aufzurufen, statt dessen können sie verwendet werden, um jegliche Methoden irgendwelcher Objekte zu schützen. Eine Konsequenz daraus ist, daß im Unterschied zu anderen Systemen, in denen die Capabilities gewöhnlich auch verwendet werden um den Empfänger der Nachricht zu ermitteln, hier kein Wissen über den Empfänger vorhanden ist. Um diese Art erweiterter Capabilities von opaquen, unstrukturierten Werten zu unterscheiden, nennen wir sie im Folgenden ein "Recht".

Askemos hat einen separaten Werteraum (spezielle Slots in Nachrichten und Plätzen), um Rechte zu verarbeiten. Dies ist eine ziemlich willkürliche Entscheidung im Interesse der Klarheit bezüglich eines so empfindlichen Themas und insbesondere motiviert durch die Möglichkeit, Rechte leicht von weniger empfindlichen Daten unterscheiden zu können. Die Bitmuster der Rechte können - im Unterschied zu einigen anderen Systemen - leicht aus ihrem Datenraum gelesen werden. Ihre Manipulation wird durch Systemrufe gesteuert.

Im Askemos werden Nachrichten als Anforderungen verstanden. Mit jeder Nachricht ist eine Menge von Rechten assoziiert, welche die Autorität definiert, mit der eine Operation ausgeführt werden soll. Wenn ein Platz auf eine Nachricht reagiert, kann durch den Programmcode festgestellt werden, ob jene Rechte ausreichend sind und dementsprechend die Operation ausgeführt oder verweigert werden.

Es muß hier festgestellt werden, daß dieses Verfahren nicht gegen schmutzige Tricks schützt, bei denen Benutzer sich einander überzeugen, ihre Rechte unabsichtlich zu verwenden.

### Die Regeln

#### Rechte Management

> "Mengentheorie kann man als exakte Form von Theologie verstehen." [Rudy

Rucker]

Ohne Superuser ist die Verwaltung von Rechten zwar etwas komplizierter, jedoch nicht viel. Am Besten versteht man jeden einzelnen Benutzer als seinen eigenen Superuser. Dies führt zu ein paar einfachen Regeln: Es existiert eine Menge $A$, welche alle Nutzer $S$ enthält.

$$S \subset A$$

Für jeden Nutzer existiert eine Menge $S$ welche (neben anderen Ressourcen zu diesem Nutzer) die Menge der Rechte $r$ dieses Nutzers enthält.

$$r \subset S$$

Jedes atomare Recht $t$ ist ein Element von $r$

$$t \in r$$

(Atomare Rechte sind den Lese-, Schreib- und Ausführbarkeitsattributen von Dateisystemen vergleichbar.)

Die Menge der Rechte, welche für eine bestimmte Operation erforderlich sind, sei $r_{needed}$.

$$r_{needed} \subseteq r$$

Jeder Nutzer $S$ kann eine Teilmenge $r_{given}$ seiner eigenen Rechte $r$ in die Menge der Rechte $r'$ eines anderen Benutzers $S'$ kopieren. Nach dieser Operation gilt:

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

Damit kein Superuser erzeugt werden kann, darf kein Benutzer alle seine Rechte vergeben, insbesondere nicht an einen einzelnen Anderen. Tatsächlich darf dies nicht einmal versehentlich möglich sein. Daraus leitet sich die Bedingung ab, daß die Menge aller kopierten Rechte $r_{copied}$ nicht der Menge aller Rechte $r$ des Benutzers entsprechen darf.

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

Jetzt ist es einfach, die Rechte zu einem Dokument durch Test auf Gleichheit zu prüfen:

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

Diese Operation ist in der Implementation zu aufwendig. Daher bleibt die Menge $r_{given}$ als Menge in $r'$ intakt. Dadurch kann ein einfacherer Test benutzt werden:

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

Diese Bedingungen finden gleichfalls Anwendung auf die Menge der Rechte $r_{given}$. Es existiert eine Obermenge von $r_{given}$, genannt $r_{owned}$, welche für das Recht steht $r_{given}$ zu löschen. Der Benutzer $S$ hält die Differenz zwischen $r_{owned}$ und $r_{given}$ zurück.

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

Mit den bis hierher angegebenen Definitionen sind die notwendigen Teile des Rechtesystems vollständig. Jeder Benutzer ist der Eigentümer eines erweiterbaren Bereiches von Rechten $r$. Individuen können untereinander Teile ihrer eigenen Rechte vergeben und entziehen. Dabei kann niemand alle Rechte eines anderen in seine Gewalt bringen.

#### Begründung und Hintergrund

Das tatsächlich implementierte System ist im Interesse der Einfachheit und Perfomance restriktiver als bislang beschrieben. Auf der anderen Seite existieren Funktionen, die den Test gängiger Fälle vereinfachen. Zukünftige Versionen können diese Regeln bis zu den im vorigen Abschnitt angegebenen Grenzen lockern.

Die Idee zu diesem Schutzschema wurde ursprünglich von dem Schema abgeleitet, welches in VSTa implementiert ist und hat sich in einem früheren Prototypen bewährt.

Es existiert jedoch ein Henne und Ei Problem: Wie werden neue Benutzer (und deren initiale Rechte) in das System eingeführt?

Benutzer verursachen Aufwand und benötigen physische Ressourcen der Maschinen, auf denen Agenten ihre Plätze unterstützen. Die physischen Maschinen sind Eigentum; konsequenter Weise sollten diese die letztliche Autorität haben, anderen Personen die Verwendung ihrer Ressourcen unter verhandelbaren Bedingungen zu gestatten. Daher kann der Eigentümer die Unterstützung für einzelne Benutzer steuern (was mittels Verträgen reguliert werden sollte).

Dies erfolgt, indem der Eigentümer Nutzerplätze zu jenem Platz binden darf, welche die Maschine selbst repräsentiert.

## Kein einzelner Ausfallpunkt, Byzantinische Protokolle in einer P2P-Welt

In einem verteilten System ist es unsinnig anzunehmen, daß alle Parteien ehrlich und zuverlässig sind. Konsequenter Weise darf es für die korrekte Funktion der virtuellen Maschine niemals nötig sein, auf eine einzelne Maschine zu vertrauen.

Ohne byzantinische Protokolle modelliert jeder Computer die Anwendung. Jeder Fehler dieses Computers und jede böswillige Manipulation wird

die Anwendung unweigerlich beeinträchtigen. Byzantinische Einigung wurde als ein Schlüsselverfahren zum Design zuverlässiger verteilter Systeme erkannt. In diesem Falle ist das Modell nur eine "lokale Annahme", eine Projektion von einem nichtphysischen Prozeß in ein physisches Modell. Stimmen die Modelle der Mehrheit der Parteien überein, so werden sie als korrekt angenommen.

Verschiedene Vorschläge und Artikel haben den Nutzen byzantinischer Protokolle demonstriert. Die meisten davon, wenn nicht alle, basieren ausschließlich auf asynchronen Nachrichten; ein weiterer Grund diese für die Definition der virtuellen Maschine zu favorisieren. Für Details zu byzantinischen Protokollen sei auf die Studie "Secure Intrusion-tolerant Replication on the Internet" [4] von C. Cachin und J. Poritz verwiesen.

Ein bedeutendes Problem mit verteilten Umgebungen ist die Gruppierung der Parteien in Koalitionen für den Prozeß der byzantinischen Einigung. Zu beachten ist insbesondere, daß die beteiligten Parteien unweigerlich Zugriff auf den Klartext des informationellen Prozesses haben. Solange kein Weg bekannt ist, wie Informationen ohne dieses Wissen verarbeitet werden können, muß der Eigentümer der Information den unterstützenden Parteien vertrauen. Eine solche Situation entspricht einem gemeinsamen Geheimnis, welches unter den ausgewählten Parteien verteilt wird, oder alternativ die Unifizierung verschiedener Geheimnisse auf höherer Ebene.

Die Bedrohung durch physische Übernahme, auch durch beliebig hohen kryptographischen Aufwand niemals kompensiert werden kann, macht es nötig die letzte Entscheidungsgewalt über die Auswahl der Parteien beim Eigentümer der Information zu belassen. Dies ist durchaus akzeptabel: sichere Verwahrung ist ein Geschäft im richtigen Leben, warum sollte das in der virtuellen Welt anders sein?

Jede Maschine die in einem Askemos Netzwerk teilnimmt, wird immer im Eigentum einer Person stehen, sei es das Standesamt einer Stadt oder der Kalender in einem Mobiltelefon. Gebühren und Steuern werden den Ersten finanzieren. Persönliches Vertrauen und Unterstützung begründen Informationen einander zu überlassen - oder eben gerade nicht. Ein weiter Bereich von Motiven begründet auf menschlicher Ebene, wie Informationen zu verteilen sind. Die Frage ist damit außerhalb des technisch Entscheidbaren.

Eine weitere, technische Motivation, die Verteilung des Vertrauens auf die menschliche Ebene zu verlagern, begründet sich in dem Fakt, daß es wesentlich einfacher ist, nicht-kooperative byzantinische Fehler zu korrigieren, denn kooperative Fehler zu tolerieren. Kooperative Fehlfunktion bedeutet gleichzeitige, böswillige Aktivität verschiedener Parteien. Eine Situation, die menschlicherseits problemlos aus der örtlichen Verteilung der Maschinen und der Reputation der Besitzer beurteilt werden kann, jedoch schwerlich "in" Computern zu entscheiden ist.

# Praxis

> "Was ist der Unterschied zwischen Theorie und Praxis? - Theoretisch gar keiner."

In der Praxis sind weitere Entscheidungen nötig. Dieser Abschnitt gibt einen Überblick über die momentane Implementation des Prototyps. Es muß bemerkt werden, daß viele der hier präsentierten Details verhandelbar sind, solange die Funktion auf übergeordneter Ebene erhalten bleibt.

Alle bislang beschriebenen Anforderungen und Aspekte sind in Code umgesetzt. Bei Entscheidungen, wie die Aspekte im Detail zu implementieren sind, wurden die besten Verfahren aus Standardtechnologien ausgewählt. Beste bedeutet dabei, daß sie theoretisch fundiert sein müssen und, sofern das mehrere Möglichkeiten zuläßt läßt, sollten alle möglich sein, jedoch wurde der erfolgreichste Weg gewählt. Weiterhin wurde darauf geachtet, so wenig Abhängigkeiten als irgend möglich einzuführen.

Das Ziel hinter dieser Entscheidungsstrategie ist es, technische, ökonomische oder soziale Einsatzhindernisse zu vermeiden, ohne die zugrunde liegende Theorie zu beeinträchtigen.

## Host System

Das derzeitige Askemos System steuert die Hardware nicht direkt. Statt dessen ist es als Server implementiert, welcher zumindest auf POSIX kompatiblen Systemen läuft, wenngleich der Großteil der von diesen gebotenen Funktionen nicht benötigt wird. Daher kann das System einfach auf jenen Servern im Internet verwendet werden, die ein vernünftiges Minimum an Sicherheit bieten. Ebenso kommt billige Hardware mit GNU/Linux und BSD basierten Maschinen in Betracht.

Der Nachteil: diese Maschinen haben immer noch einen Superuser. Für hochsichere Qualitätssysteme sollten keine weiteren Dienste auf der Askemos Servermaschine laufen und der Superuseraccount sowie andere Mechanismen zur Ausführung beliebiger Programme wie `su` sollten außer Funktion gesetzt werden.

Die Größe und Komplexität des gesamten Systems lassen es zu, Askemos auf Mobiltelephone, Handhelds und ähnliche Hardware zu portieren.

### Programmiersprache

Wie bereits erwähnt ist es machbar, jede beliebige Sprache zur Entwicklung von Anwendungen bereit zu stellen. Dies muß lediglich auf eine solche Art und Weise erfolgen, daß das Programm in einer isolierten Umgebung eingeschlossen werden kann. Dies kann für einige Sprachen einen erheblichen Aufwand bedeuten.

Die für den Prototyp verwendete Sprache, Scheme, welche derzeit auch für die Implementation verwendet wird, wurde gewählt, weil es eine einfache und kleine Sprache ist, die Mittel für sichere Berechnungen (Bereichstest, keine Zeiger) sowie leistungsfähige Abstraktionsmechanismen (Funktionen höherer Ordnung und Continuations) hat. Scheme ist gut standardisiert und der Standardisierungsprozess wird fortgeführt. Es existieren viele Implementationen für alle Arten von Umgebungen bis hinunter zu Handheld Hardware, die Java sowie die .NET-Plattform. Manche Compiler sind frei verfügbar. Sie erzeugen effektiven C-Code und die Verbindung mit herkömmlichen Systemen ist gewöhnlich einfach. Zusammenfassend kann gesagt werden: Scheme ist preiswert und es ist immer möglich, damit auf andere Plattformen zu migrieren.

Weiterhin ist Scheme ein direkter Vorläufer des DSSSL (ISO 10179:1996) Standard, welche wiederum die Entwicklung der XSLT Recommendation des W3C beeinflußt hat. Letztere sind beides rein funktionale Sprachen und passen daher hervorragend zu Askemos' virtuellen Maschine. Scheme, als der Vorgänger von DSSSL und XSLT, machte es leicht, diese Sprachen auf Anwendungsebene bereit zu stellen.

Die Gefahr Fehler zu machen wurde erheblich vermindert, indem der Askemos Kern selbst fast vollständig in funktionalem Stil programmiert wurde. Dies macht es auch leicht zu garantieren, daß Plätze auf ihren Datenbereich beschränkt sind. Es ist nicht notwendig zu diesem Zweck Adressräume des Hostsystems zu verwenden.

Die gewählte Implementation, RScheme, bietet weiterhin preemptive Threads (notwendig für das asynchrone Verarbeitungsmodell).

### Datenmodell und Kompatibilität

Weder die Askemos virtuelle Maschine noch die Implementation sind an ein spezifisches Datenmodell gebunden. Derzeit konvergiert der Markt zu XML als universellem Datenaustauschformat. XML kann alle Arten strukturierter Daten darstellen, ist gut standardisiert und weit verbreitet. Daher wurde XML als "natürliches" Datenformat der Implementation ausgewählt; es wurde auf die Verarbeitung von XML hin optimiert.

Sich auf XML zu einigen, räumt auch die meisten Kompatibilitätsprobleme aus dem Weg, da mehr und mehr althergebrachte Systeme XML-Kodierung zum Datenaustausch anbieten.

Die Daten werden derzeit in zwei Speichermedien gehalten: a) eine persistente Datenbank, welche die Baumstrukturen, Hashtabellen etc in ihrer internen Form speichert. Die Technik wird "pointer swizzling at page fault time" genannt. Und b) in einer Dateisystemrepräsentation.

### Netzwerkprotokolle

Die bedeutendsten Anwendungsprotokolle im Internet sind HTTP und SMTP. Die derzeitige Implementation benötigt nur diese Protokolle und kann damit zu den meisten Anwendungen verbunden werden, welche zur zwischenmenschlichen Kommunikation verwendet werden.

Aufgrund der Bedeutung dieser Protokolle erlauben die meisten Firewalls diese Verkehrsart, so daß keine Probleme aus dieser Richtung bestehen. Wenngleich SMTP zur Abstimmung in byzantinischen Protokollen wahrscheinlich dazu führen würde, daß das System praktisch stehen bleibt.

Alle Platz zu Platz Kommunikation wird konzeptionell als SOAP-Nachricht verstanden. Konzeptionell meint hier, daß verschiedene, redundante Informationen (envelope und header) solange nicht generiert werden, bis diese tatsächlich benötigt werden.

### Rechtliche und Langzeitverfügbarkeit

Alle Datenformate, Programmiersprachen und andere Standards, welche für Askemos verwendet wurden, sind frei von Forderungen aus "intellectual property". Niemand kann durch plötzliche Forderungen die Benutzbarkeit des Systems in Frage stellen. Die Implementation selbst sowie die darunter liegende Sprache sind unter der GNU Public License[9] verfügbar.

## Privacy

Im Laufe der letzten Jahre wurden verschiedene Versuche gemacht, globale Benutzeridentifizierung und Datenspeicher zu verwirklichen. Diese scheinen eine Voraussetzung für den elektronischen Handel zu sein und würden viele administrative Aufgaben vereinfachen. Wenngleich kein abschließendes Urteil über deren Erfolg möglich ist, so scheinen diese Systeme auf erheblichen Widerstand seitens möglicher Kunden zu stoßen. Daher wurden solche Systeme immer wieder zurückgestellt.

Besagter Widerstand stammt offenbar aus dem menschlichen Interesse nach informationeller Selbstbestimmung. Die Ansammlung von Nutzerdaten wie sie die vorgeschlagenen Systemen verursachen, treiben jedoch nicht nur paranoide Sicherheitsexperten zum Wahnsinn, sondern werden auch von gewöhnlichen Benutzern gefühlt. Askemos als verteiltes System ohne Superuser kann den selben Nutzen erreichen, ohne Sicherheit, Verfügbarkeit und informationelle Selbstbestimmung zu gefährden. Es muß lediglich durch ein geeignetes Vertragssystem vervollständigt werden, um die Qualität der Dienste zu sichern. Dann sind die Nutzer frei zu entscheiden, wem sie bezüglich Ihrer Informationen vertrauen.

## Zusammenfassung

Das Askemos System wurde als praktisch einsetzbares Betriebssystem dargestellt. Seine Komponenten sind vollständig und gut untersucht, keine basiert auf unvollständigen und ungeprüften Annahmen, auch interferieren die Komponenten nicht. Zur Definition wurden ausschließlich öffentliche Standards verwendet. Alle Komponenten sind verfügbar und frei von überraschenden Forderungen, eine Voraussetzung die "digitale Teilung" zu vermeiden.

## Literatur


[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/c̃ca/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] "Petrinet" `http://www.daimi.au.dk/PetriNets`

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