Telescript

Carsten Kübler

HS Mobile Agenten, WS 1996/97*

*1997 ist vom Großforschungsinstitut SRI International in Stanford, Kalifornien USA, zum **„Jahr der Agenten“** erklärt worden. [DieZeit97]

Inhaltsverzeichnis

1 General Magic

1990 begann Apple mit der Entwicklung von Plänen für „Personal Communications“. Als in diesem Jahr klar wurde, daß nicht eine einzelne Firma alle Komponenten ausliefern kann, wurde General Magic im Mai 1990 gegründet.

In der Zwischenzeit hat sich General Magic vergrößert.

- Die Einnahmen beliefen sich 1995 auf $14,165,000.
- Es sind weltweit 207 Mitarbeiter angestellt.
- Es gibt Büros in Sunnyvale (Kalifornien USA), Columbia (Süd-Carolina USA), Paris (Frankreich) und Tokyo (Japan).
- Produkte sind Tabriz Agent Products, Magic Cap und weitere Kommunikationssoftware.

Die Telescript Technologie wird auch kommerziell genutzt. Sie wird z.B. eingesetzt

in Produkten (Magic Cap): *PIC-1000* und *PIC-2000* von Sony, *Envoy 150* von Motorola, usw.; Magic Cap wird auch für Windows95 angeboten.

für Dienste (Telescript): *PersonaLink* von AT&T (von 1994 bis 30. August 1996), *Paseo* von NTT (seit März 1996), *Pyramide* von France Telecom (in der Entwicklung).

Es gibt eine General Magic Allianz, der viele größere Firmen angehören. Jeder strategische Geschäftspartner investiert in die Gesellschaft. Heute ist die Allianz stärker als je zuvor, da sie aus mehr als zwei Dutzend Partnern besteht. Darunter befinden sich die weltweit größten und erfolgreichsten Unternehmen, z.B. Apple, AT&T, Cable & Wireless, France Telecom, Fujitsu, Matsushita, Mitsubishi, Motorola, Northern Telecom, NTT, Oki, Philips, PTT Telecom, Sanyo, Sony, Toshiba, usw. Diese Allianzpartner treffen sich formell zweimal im Jahr und diskutieren über die Entwicklungen der Märkte und der Technologien.

Weitere Informationen gibt es von [GenMagic], [Sony] und [mobilis].

2 Telescript Technologie

Neue Kommunikationsparadigmen benötigen neue Kommunikationstechniken. Die Technik des mobilen Agenten soll auf einer Vielzahl heutiger und kommender Computer und Kommunikationsmedien einsetzbar sein. Deshalb wurde die Telescript Technologie entwickelt, die viele Konzepte beinhaltet.

Die Telescript Technologie besteht aus

1. dem Telescript Modell.
2. der Sprache Telescript, in der Plätze und Agenten programmiert werden. Sie ist die Implementierung des Modells.
3. einer Implementierung der Sprache durch einen Interpreter, der durch die Telescript Engine realisiert wurde.
4. der Telescript Welt Kommunikationsprotokollen, die es Engines auf verschiedenen Computern erlauben, Agenten zur Erfüllung der ***go*** Operation auszutauschen.

2.1 Telescript Modell

Das Telescript Modell ist ein Modell von mobilen Agenten, in dem Agenten von einem Platz zu einem anderen reisen können. Die Mächtigkeit, die durch die Mobilität von Agenten entsteht, wird durch Permits (siehe Abschnitt 3.4.2) eingeschränkt. Permits kann ein Programmierer oder Administrator eines Telescript Netzwerkes nutzen, um einzelne Fähigkeiten einem einzelnen Agenten oder Platz zu speziellen Anlässen zu gestatten.

Die Autorität, in der physikalischen Welt, eines Platzes oder Agenten wird durch seinen *telename* offenbart, welcher weder verfälscht noch vor einem anderen Platz oder Agenten zurückgehalten werden kann. Ein Platz hat weiterhin eine *teleaddress*, die seine Lokation in der elektronischen Welt (siehe Abschnitt 2.4) und die Autorität, auf dessen Computer der Platz existiert, offenbart.

Jeder Platz wird typischerweise permanent von einem Agenten[1] mit der Autorität des Platzes bewohnt und von anderen Agenten zeitweilig besucht. Der typische Agent reist, um einen entfernt angebotenen Dienst zu erreichen. Plätze können geschachtelt werden.

2.2 Telescript Sprache

Die Telescript Sprache hat folgende Eigenschaften, die die Entwicklung einer Kommunikationsanwendung erleichtern: [White95a]

Objektorientiert: Die Sprache behandelt jedes Stück Information wie in SmallTalk als Objekt. Es gibt vordefinierte Klassen grundlegender Natur, z.B. *Agent*, und von Entwicklern definierte (Sub)Klassen, z.B. *Einkaufsagent*.

Dynamisch: Ein Agent kann ein Informationsobjekt von einem Platz, in einem Computer, zu einem anderen Platz, in einem anderen Computer, mitnehmen, auch wenn die Klasse des Objektes beim Ziel unbekannt ist, da die Klasse mitübertragen wird.

Persistent: Wohin ein Agent geht, er und seine Daten (incl. Program Counter) werden sicher in nichtflüchtigem Speicher gesichert. Dies läßt einen Agenten einen Engine– oder Computerabsturz überleben. Dies gilt auch für Plätze und Objekte.

Portabel und sicher: Ein Computer führt einen Agenten nicht direkt sondern durch eine Engine aus. Dadurch hat ein Agent keinen direkten Zugriff auf Prozessor, Speicher, Dateisystem oder Peripheriegeräte.

Kommunikationszentriert: Bestimmte Operationen der Sprache lassen einen Agenten komplexe Netzwerkaktionen wie z.B. Transport, Navigation, Authentifizierung, Zugriffskontrolle, usw. durchführen.

Die Telescript Sprache hat weitere Merkmale:

- Jeder Agent bzw. Platz ist ein eigener Prozeß. Bis jetzt sind die Klassen *Agent* und *Place* die einzigen Subklassen der Klasse *Process*.

[1] Der Agent ist häufig erforderlich, so daß das Modell die Funktionalität des Agenten in den Platz aufgenommen hat.

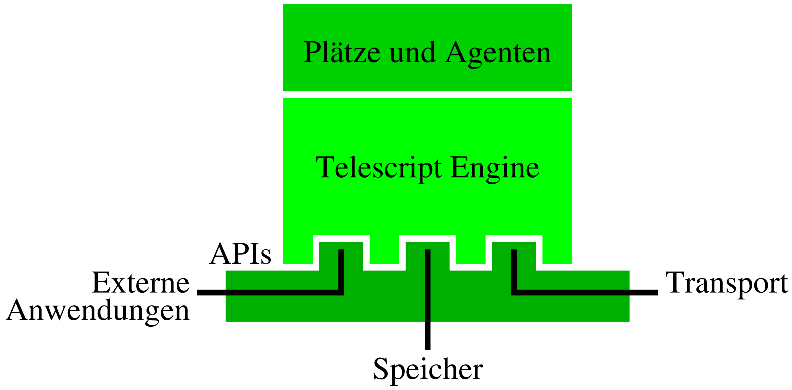
- Jedes Objekt wird von einem Objekt besessen. Plätze und Agenten gehören sich selbst. Jedes andere Objekt gehört entweder einem Platz oder Agenten. Ein Besitzer eines Objektes kann dieses isolieren, so daß alle Referenzen anderer Objekte, die einem anderen Besitzer gehören, gelöscht werden.
- Eine Operation kann im Falle eines Fehlers eine Exception auslösen (siehe Abschnitt 3.9).
- Es gibt eine eingeschränkte Form der Mehrfachvererbung (Mixin Klassen) und generische Klassen.
- Eine Klasse oder ein Feature, d.h. eine Operation oder Attribut, kann versiegelt werden. Diese Klasse kann nicht geerbt bzw. das Feature kann in einer Subklasse nicht überschrieben werden.
- Es gibt keine Zeiger. Dafür gibt es zwei Arten, geschützte und ungeschützte, Referenzen. Operationen, die über eine geschützte Referenz ausgeführt werden, können das Objekt nicht verändern.
- Jeder Platz bzw. Agent besitzt Permits (siehe Abschnitt 3.4.2). Diese geben ihm Rechte und schränken ihn ein.
- Es gibt gesponserte Operationen. Werden diese Operationen aufgerufen, werden sie unter den Permits und der Autorität des Besitzers dieses Objektes ausgeführt und nicht unter den Permits und der Autorität des Aufrufers der Operation. Eine Engine ruft immer gesponserte Operationen eines Prozesses auf.
- Die Sprache besitzt mächtige Klassen wie *List*, *Directory*, usw.

Normalerweise wird jedes Programm in High–Telescript geschrieben (ähnlich C++ oder Pascal) und mit einem Compiler in Low–Telescript (ähnlich Postscript), die stack-orientiert ist, umgewandelt.

2.3 Telescript Engine

General Magic hat zwei Implementierungen der Sprache Telescript entwickelt. Eine Engine ist in Magic Cap PICs eingebaut und wurde für eine neue Kategorie von Kommunikationsgeräten entwickelt, d.h. Magic Cap ist eine einfach bedienbare Kommunikationssoftwareplattform, die viele Kommunikationsmittel, z.B. E–mail, Pager, Telefon, Fax, usw., verbindet. Die zweite Engine ist zur Konstruktion öffentlicher Dienste und Server in Unternehmen entworfen worden und wurde wegen der Portabiliät in C++ geschrieben.

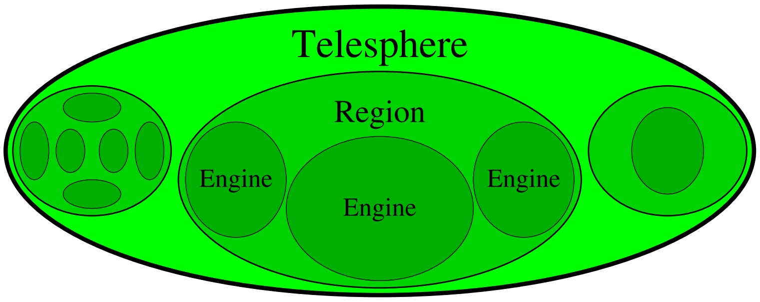
Die Engine ist die Implementierung der Telescript Sprache und interpretiert Low–Telescript. Eine Engine baut auf drei Anwendungprogrammschnittstellen (APIs) auf.



Über die Speicher–API hat die Engine Zugriff auf den nichtflüchtigen Speicher, z.B. Festplatte. Über die Transport–API kann die Engine auf die Kommunikationsmedien zugreifen, die für den Transport der Agenten zu bzw. von einer anderen Engine nötig sind. Über die externe Anwendungs–API können Teile der Anwendung, die z.B. in C++ implementiert wurden, mit den Teilen, die in Telescript geschrieben sind, interagieren.

2.4 Kommunikation zwischen Engines in der Telesphere

Die elektronische Welt von Telescript, die Telesphere, ist in eine oder mehrere Regionen unterteilt. Jede Region wird von einer anderen Autorität betrieben, die für sie verantwortlich ist. Eine Autorität kann eine Person, z.B. Benutzer eines PICs, oder eine Organisation, z.B. NTT, sein. Jede Region besteht aus einer oder mehreren Engines, damit die nötige Leistung erbracht werden kann.



Ein Agent kann zu verschiedenen Plätzen in unterschiedlichen Engines, die miteinander verbunden sind, reisen; gleich ob die Plätze in derselben Engine, derselben Region oder in verschiedenen Regionen sind.

Agenten können mit der ***go*** Operation von einer Engine zu einer anderen reisen.

Das Telescript *Plattformverbindungsprotokoll* spezifiziert, wie ein verschlüsselter Agent zwischen Engines ausgetauscht wird. Es ist auch beschrieben, wie sich Engines authentifizieren, z.B. durch den Einsatz von Kryptographie mit öffentlichen Schlüsseln.

3 Aspekte von Telescript für mobile Agenten

Diese Übersicht beschreibt die Möglichkeiten der Telescript Sprache und die Fähigkeiten von Tabriz AgentWare mit Telescript Developer Environment (TDE) in der Alphaversion. AgentWare besitzt eine HTML–API zu Standard–Webservern. Die zusätzlichen Fähigkeiten von Magic Cap werden nicht betrachtet.

3.1 Kommunikation

In der Referenzsprache gibt es nur lokale Kommunikation, aber es soll in einer zukünftigen Version auch globale Kommunikation möglich sein.

Es gibt zwei vorgesehene Arten, wie Prozesse miteinander kommunizieren können:

1. Wenn ein Prozeß eine Referenz auf einen anderen besitzt, kann er mit dem anderen mit objektorientierten Techniken[2] kommunizieren. Z.B. kann ein Agent eine Operation eines Platzes, den er gerade besucht, aufrufen. Diese Operation kann ein Objekt als Ergebnis liefern.
2. Events können zur Synchronisation eingesetzt werden, da aber ein Event ein Objekt einer (Sub-)Klasse *Event* ist, kann dieses Objekt zusätzliche Nachrichten enthalten. Ein Broadcastevent kann z.B. ein Objekt beinhalten, das an eine eingeschränkte Gruppe gesendet werden kann. Die Gruppe beschränkt sich maximal auf den Platz, den der Prozeß (der das Event auslöst) gerade besucht und alle Prozesse innerhalb dieses Platzes incl. deren Prozesse (und so rekursiv weiter).
3. Über die HTML–API kann ein Prozeß Seiten auf einem Webserver anbieten und unter Angabe einer URL Seiten anfordern. Eine angeforderte Seite kann in Telescriptobjekte umgewandelt und damit einfacher ausgewertet werden. Dadurch ließe sich globale Kommunikation über Umwege realisieren.

Es ist weiterhin eine ***connect*** Operation geplant, mit der es möglich sein soll, daß Agenten bzw. Plätze auf verschiedenen Computern direkt kommunizieren können.

Engines von Tabriz AgentWare kommunizieren über TCP oder UDP miteinander.

Die Kommunikation zwischen Telescript– und Nichttelescriptprogrammen wird in Abschnitt 3.5 beschrieben.

3.2 Namen und Adressen

Jeder Prozeß besitzt einen Namen (*Telename*), der ihn von jedem anderen Prozeß unterscheidet. Der Name besteht aus einem Namen der Autorität, die für das Handeln des Prozesses verantwortlich ist, und einer Identität, die jeden Prozeß einer Autorität unterscheidet. Eine Autorität kann eine Person oder eine Organisation sein. Jeder Platz besitzt weiterhin eine Adresse (*Teleaddress*). Alle Plätze, die bekanntgegeben werden, haben verschiedene Adressen, die den Ort des Platzes in der Telesphere festlegen. Die Adresse besteht aus dem Namen der Autorität, die für die Telescript Region verantwortlich ist, und einer regionalen Bezeichnung, die alle bekanntgegebenen Plätze unterscheidet. Die Namen aller Autoritäten und alle Identitäten sind Oktettzeichenketten, die für einen Menschen keine Bedeutung haben und durch einen von General Magic empfohlenen mathematischen Algorithmus berechnet werden. Dieser Algorithmus wurde nirgends beschrieben. Jede regionale Bezeichnung eines Platzes ist ebenfalls eine Oktettzeichenkette, welche aber durch einen Algorithmus, der regionalspezifisch sein kann, berechnet wird.

Ein Prozeß kann seinen *Telename* einem anderen nicht vorenthalten oder verfälschen, so gibt es keine anonymen Prozesse. Das gleiche gilt für die *Teleaddress* eines Platzes.

Ein spezieller Dienst kann von einer Instanz einer Subklasse einer allgemeinen Klasse oder einer Mixin Klasse erbracht werden, indem Operationen, die den Dienst realisieren, aufgerufen werden. Der Name einer Klasse ist ebenfalls eine Oktettzeichenkette, die für einen Menschen keine Bedeutung hat.

Meist gibt es mehrere Wege, mit einer Person bzw. einem Unternehmen Kontakt aufzunehmen und diese können mehrere Telescript Anwendungen betreiben oder benutzen. Die TDE Klasse *AddressCard* kann viele Informationen in einer Adreßkarte bündeln.

[2] Methoden werden in Telescript normalerweise als Operationen bezeichnet.

Dazu gehören der Name der Person bzw. des Unternehmens, alle Adressen (E–mail–, Fax–, Pager–, Postadresse, usw.) und Telescript Anwendungen. Die Plätze der Anwendungen werden mit einer *Facility* Liste verwaltet. Für jede Art von Anwendung wird ein *facility* benutzt, womit ein Name (eine Bezeichnung für die Art der Anwendung) mit einem Ticket oder einer Ticketliste assoziiert wird. Das Ticket enthält alle Informationen, die ein Agent benötigt, um zu einem gewünschten Platz reisen zu können.

Ein Platz kann sich und einen Teil seiner Operationen (AgentWare Klassenbibliothek) bei einem Platz *Registry* registrieren lassen. Jedem registrierten Platz bzw. Operation wird ein Name gegeben. Den Platz *Registry* kann man über einen Standard–Webserver und Agenten ansprechen. Damit ist es möglich, durch eine URL Adresse Operationen eines Platzes aufzurufen, da ein Teil der URL den Platz und die gewünschte Operation über die registrierten Namen spezifiziert.

3.3 Migration

Die Migration eines Agenten ist transparent, d.h. das Programm, die Daten (alle Objekte, die der Agent besitzt) und der Ausführungszustand migrieren zusammen. Damit ein Agent von einem Platz zu einem anderen migrieren kann, muß der Agent seine ***go*** Operation aufrufen. Er kann auch Kopien von sich zu verschiedenen Plätzen migrieren lassen, indem er die ***send*** Operation aufruft. Das Ziel wird durch ein *ticket* spezifiziert. Beide Operationen sind versiegelte Operationen der Klasse *Agent*.

Bevor ein Agent einen Platz betritt, wird der Platz automatisch von der Engine um Erlaubnis gefragt, ob der Agent den Platz betreten darf, indem die gesponserte Operation ***entering*** des Platzes aufgerufen wird. Dies ist eine Firewall für diesen Platz. Die Operation ***entering*** erhält den Namen des Agenten, den Namen seiner Klasse, seine gewünschten Permits und sein Ticket. Sie erhält jedoch keine Referenz auf den Agenten, d.h. der Platz kann keine Operationen eines Agenten aufrufen, bevor er nicht den Platz betreten hat. Die Operation kann die vom Agenten gewünschten Permits einschränken (Beispiel in Abschnitt 5.2.2 auf Seite 33 Zeile 179-208).

3.4 Sicherheit

In der Sprache Telescript gibt es viele Sicherheitsmechanismen. Einen tieferen Einblick erhält man in [SaveSec96]. Auch bei Telescript muß ein Benutzer dem Betreiber einer Telescript Plattform vertrauen, daß seine Daten nicht willkürlich freigegeben werden.

3.4.1 Laufzeitsicherheit

Wegen der interpretierten Natur kann Telescript Laufzeitsicherheit unterstützen und den Programmierer von Zufällen abschirmen, die eine Engine zum Absturz bringen könnten. Außerdem ist jedes Informationsobjekt persistent. Es gibt weder Pointer noch Pointerarithmetik. Jede Engine unterstützt eine Laufzeittypüberprüfung, Garbage Collection und Exceptions. Die Telescript Sprache definiert eine Anzahl an eingebauten Klassen, die eine Subklasse von *Collection* sind. Durch sie kann nur auf ihre Objekte über „sichere“ Operationen zugegriffen werden; z.B. erlaubt *List* einen Zugriff auf seine Elemente über einen Index. Ist der Index unerlaubt, wird eine Exception ausgelöst.

3.4.2 Prozeßsicherheit

Grundsätzlich werden alle Interprozeßzugriffe durch die Telescript Engine vermittelt. Jede Engine unterstützt

- eine unfälschbare Authentifizierung jedes Prozesses (siehe Abschnitt 3.2).
- geschützte Referenzen.
- Schutz durch Einkapselung privater Properties und Features. Mit öffentlichen Operationen kann man eine eigene Zugriffskontrolle auf private Features realisieren (Beispiel in Abschnitt 5.2.1 auf Seite 17 Zeile 46-81).

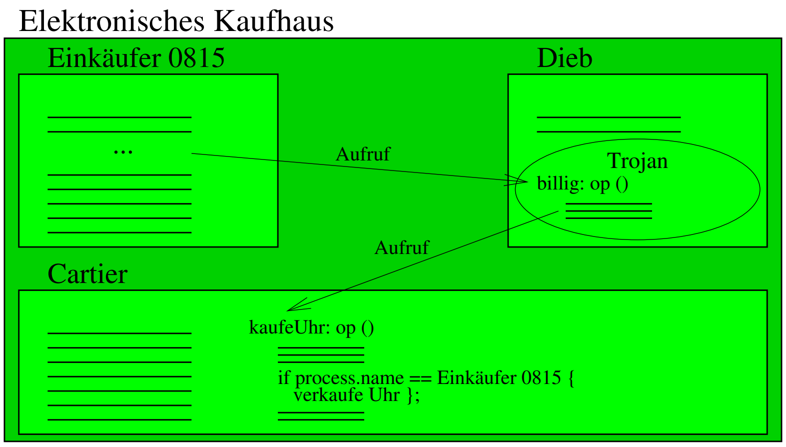
  Telescript bietet verschiedene Wege, die Autorität und Klasse des Aufrufers einer Operation zu identifizieren. Von der Engine werden dafür direkt globale Variablen angeboten (Beispiele in Abschnitt 5.2.1 auf Seite 17 Zeile 46-81 und in Abschnitt 5.2.2 auf Seite 34 Zeile 231-239).

  Unter anderem gibt es

  - den aktuellen Prozeß *process*. Dies ist der Prozeß, dem die gesponserte Operation gehört, die am weitesten entfernt die aktuelle Operation einschließt. Die Referenz ist ungeschützt.
  - den aktuellen Besitzer *owner*. Dies ist der Prozeß, dem ein neu erzeugtes Objekt gehört. Z.B. kann der Besitzer eines Objektes zeitweilig geändert werden, indem im Programm ein Telescript **own** Block benutzt wird. Jedes Objekt, das in diesem Block erzeugt wird, gehört dem Prozeß, dem das Objekt gehört, in dessen Operation der **own** Block ist. Die Referenz ist auch ungeschützt.
  - den aktuellen Sponsor *sponsor*. Dies ist der Prozeß, dem die gesponserte Operation gehört, die am nahesten die aktuelle Operation einschließt, d.h. unter dessen Autorität und Permits diese Operation ausgeführt wird. Die Referenz ist ungeschützt.
  - den Aufrufer *client*. Dies ist das Objekt, das die aktuelle Operation aufgerufen hat. Die Referenz ist ungeschützt bzw. geschützt, je nachdem ob die Referenz von *client* auf dieses Objekt ungeschützt bzw. geschützt ist.

  Die globalen Variablen können alle auf unterschiedliche Objekte referenzieren.

  Ein Beispiel soll zeigen, weshalb verschiedene globale Variablen angeboten werden.



Ein *Einkäufer 0815* und ein *Dieb* (beides Agenten) besuchen ein *Elektronisches Kaufhaus*, in dem es einen Laden *Cartier* (beides Plätze) gibt. Der *Einkäufer 0815* trifft sich mit einem Agenten (z.B. *Dieb*), der ihn beraten soll. Von ihm erhält er eine Referenz auf ein Objekt (z.B. der Klasse *Trojan*), das ihm helfen soll. Der naive *Einkäufer 0815* ruft die Operation ***billig*** auf, die etwas (vom *Einkäufer 0815*) Ungewolltes ausführt (z.B. ***kaufeUhr***). *Cartier* überprüft nur die Identität des Prozesses und sieht, daß der aktuelle *process* die Autorität *Einkäufer 0815* hat, der vor langer Zeit Stammkunde war und verkauft ihm eine Uhr, die der *Dieb* entgegennimmt und „mit ihr“ verschwindet.

Hätte beispielsweise *Cartier* die Autorität des Besitzers des *client* überprüft, wäre der *Dieb* enttarnt worden oder wäre *Cartier* vorsichtiger gewesen und hätte getestet, ob er den aktuellen Sponsor kennt und der *Einkäufer 0815* nur eine gesponserte Operation ***billig*** des „unbekannten“ Objektes aufgerufen, hätte der *Dieb* keine Chance gehabt.

- Permits. Permits bieten einen Mechanismus, um den Ressourcenverbrauch zu limitieren und die Fähigkeiten des ausgeführten Programmes zu kontrollieren. Mit *Permit* kann man z.B. das maximale Alter und die maximale Größe angeben sowie ob ein Prozeß neue Prozesse erzeugen kann, ob die ***go*** Operation aufgerufen werden darf bzw. ob der Prozeß die Permits anderer Prozesse erhöhen oder erniedrigen kann.

  Es gibt vier Arten von Permits:

  1. *Angeborene* Permits werden vom Erzeuger des Prozesses zugewiesen (Beispiele in Abschnitt 5.2.1 auf Seite 20 Zeile 323, auf Seite 30 Zeile 1082-1089 und in Abschnitt 5.2.2 auf Seite 36 Zeile 327-329).
  2. *Lokale* Permits können von einem Platz einem Agenten, der den Platz betritt, oder einem Prozeß, der innerhalb des Platzes erzeugt wird, auferlegt werden. Diese Permits werden nur innerhalb des Platzes angewandt (Beispiel in Abschnitt 5.2.2 auf Seite 33 Zeile 186-192).
  3. *Regionale* Permits sind wie *local* Permits, aber sie werden von dem Engineplatz auferlegt. Diese Permits werden nur innerhalb einer oder einer Menge Engine(s) angewandt. Ein Engineplatz ist der äußerste Platz einer Engine. Alle Plätze in dieser Engine sind Subplätze dieses Engineplatzes.
  4. *Zeitweilige* Permits werden einem Telescript **restrict** Block auferlegt (Beispiel in Abschnitt 5.2.1 auf Seite 25 Zeile 782-787).

  Die effektiven Permits eines Prozesses werden durch den logischen Schnitt aller vier Arten berechnet.

  Zeitweilige Permits können den Schaden eines zweifelhaften Code beschränken. Z.B. kann die Ausführungszeit eines Code eingeschränkt werden, so daß der Agent noch Zeit hat, seinen Zielplatz zu erreichen (Beispiel in Abschnitt 5.2.1 auf Seite 24 Zeile 755-761, 777-791).

- sicherheitsorientierte Mixin Klassen. Es gibt

  - *Unmoved.* Ein Agent kann solch ein Objekt nicht mit sich nehmen, wenn er ***go*** aufruft. Z.B. sind Plätze unbeweglich.
  - *Uncopied.* Ein Versuch, solch ein Objekt zu kopieren, gibt eine Referenz auf das Original zurück.
  - *Copyrighted.* Die Initialisierung eines solches Objektes schlägt fehl, wenn nichtautorisiert versucht wird, dieses Objekt zu erzeugen.

  - ▷ *Protected.* Solch ein Objekt kann nach der Initialisierung nicht mehr verändert werden und alle Referenzen auf dieses Objekt verhalten sich wie geschützte Referenzen.

- ▶ vermittelnde Protokolle für Prozeßrendezvous.

  Bevor ein Agent einen Platz betreten oder ein neuer Prozeß im Platz erzeugt werden darf, gibt die Engine diesem Platz die Gelegenheit, das Betreten zu verweigern (siehe Abschnitt 3.3).

  Analog zum Betreten eines Platzes gibt es ein Protokoll, wie zwei Agenten, die die Mixin Klasse *MeetingAgent* besitzen, die Gelegenheit bekommen, ein Treffen anzunehmen oder abzulehnen. *Meeting* Agenten können sich nur in Plätzen mit der Mixin Klasse *MeetingPlace* treffen (siehe Abschnitt 3.7).

3.4.3 Systemsicherheit

TDE bietet Klassen an, die Programmen den Zugriff auf darunterliegende Systemressourcen erlauben. Sie sind für jeden Prozeß sichtbar, aber ihr Zugriff wird mit den Mixin Klassen *Copyrighted*, *Uncopied* und *Unmoved* kontrolliert.

Im allgemeinen sollte der direkte Zugriff nur auserwählten Prozessen erlaubt werden, die unzuverlässigen Prozessen Dienste anbieten.

3.4.4 Netzwerksicherheit

Agenten können zwischen Regionen über „sichere Kanäle“ übertragen werden. Es gibt einen Authentifizierungsmechanismus von Engines fremder Regionen. Die bislang eingesetzte Technik soll noch verbessert werden.

3.5 Sprachen

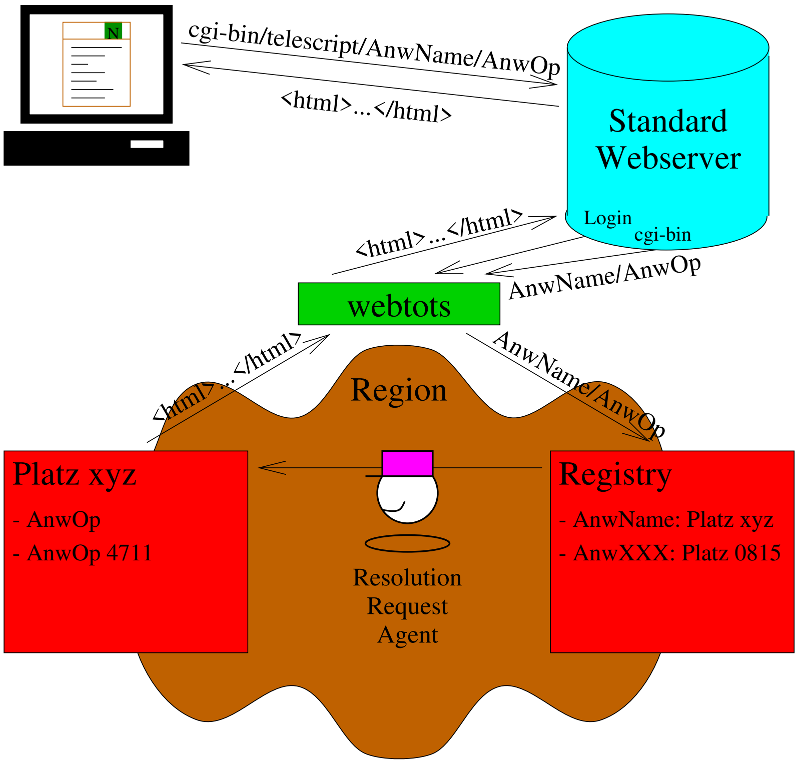
Alle Agenten und Plätze können nur in der Sprache Telescript geschrieben werden. Eine Anwendung muß nicht ausschließlich in Telescript programmiert werden sondern kann auch Teile in Java, C++ und C beinhalten. Telescript kann Java Applets in HTML Seiten aufrufen und Applets können über einen Webserver Telescript Operationen über die AgentWare HTML-API aufrufen. Damit kann man die Stärken beider Sprachen nutzen. Es soll auch eine Möglichkeit geben, mit Java Applets über Java Sockets zu kommunizieren, aber das in [DevGuide96] zitierte Beispiel war bei der ausgelieferten Software nicht enthalten. Jede Engine hat eine externe Anwendungs–API (siehe Abschnitt 2.3) über die Telescriptprogramme mit externen Programmen kommunizieren können. In der TDE gibt es eine Sammlung von Klassen für die Kommunikation von Programmteilen über die Anwendungs–API, das External Application Framework (EAF). Das EAF besteht aus Telescript– und C++ Klassen, mit denen Telescriptprogramme mit C++ bzw. C Programmen kommunizieren können. Die Ausführung eines C++ bzw. C Programmes wird als externer Prozeß bezeichnet. Die Kommunikation zwischen einem externen Prozeß und einer Telescript Engine geschieht über eine TCP-Verbindung. Die EAF kann für stationäre Anwendungen eingesetzt werden.

3.6 Interaktion mit dem Benutzer

Wie in Abschnitt 3.5 beschrieben wurde, kann ein Telescriptprogramm durch die EAF mit externen Programmen kommunizieren. Die Mächtigkeit dieser Programme kann durch Telescript nicht eingeschränkt werden und deshalb können die externen Programme mit allen Mitteln, die ihnen zur Verfügung stehen, einem Benutzer Möglichkeiten zur Interaktion anbieten.

Weiterhin werden in der TDE Klassen angeboten, mit denen ein GUI erzeugt werden kann und der Benutzer mit einem Telescriptprogramm interagieren kann.

Mit einem beliebigen Webbrowser kann ein Benutzer (mit Hilfe eines Standard–Webserver) über die HTML–API mit Telescriptprogrammen interagieren.



3.7 Auffinden von Plätzen bzw. Agenten und deren Diensten

Besitzt man ein Ticket für einen gewünschten Platz, so kann man mit ***go*** bzw. ***send*** zu dem im Ticket spezifizierten Platz reisen.

Ein Ticket kann einen Platz durch einen *telename*, eine *teleaddress*, einen Namen einer Klasse und Routinginformationen (z.B. Internetadresse eines Computers und eine Portnummer) spezifizieren oder nur durch einen Teil dieser Informationen.

Ein Platz kann Pakete veröffentlichen. Ein Paket verknüpft den *telename* des Platzes mit einem Telescript Objekt. Ein Platz hat mehrere Möglichkeiten, nach einem Objekt oder *telename* in den veröffentlichten Paketen einer Engine zu suchen. Die Pakete werden vielseitig, z.B. zum Auffinden von Plätzen in einer Engine, verwendet.

Wenn man keine Informationen für ein Ticket besitzt, so gibt es bei TDE eine Möglichkeit. Jede Engine kann auch eine Adreßkarte (siehe Abschnitt 3.2) besitzen. In dieser Karte gibt es eine *Facility* Liste, die alle *Facility* Listen der Adreßkarten jeder Person bzw. jedes Unternehmens enthält, sofern diese von der Person bzw. dem Unternehmen auf dieser Engine veröffentlicht wurde. (Hierfür werden veröffentlichte Pakete eingesetzt.) Somit kann man über den Namen, d.h. den Bezeichner einer Art von Anwendung, in dieser *Facility* Liste das entsprechende *facility* mit allen veröffentlichten Tickets erhalten (Beispiel: „Aufnahme des Dateiservers in die Adreßkarte des Betreibers des Servers als *facility*" in Abschnitt 5.2.2 auf Seite 32 Zeile 139-159 und Beispiel: „Aufnahme aller *facility* aus einer Adreßkarte in die Adreßkarte der Engine" in Abschnitt 5.2.2 auf Seite 36 Zeile 327-336 und Beispiel: „Erhalt der *facility* der Dateiserver einschließlich Tickets und Besuch aller Server" in Abschnitt 5.2.1 auf Seite 24 Zeile 742-749 und Zeile 764-803).

In Abschnitt 3.2 wird beschrieben, daß Plätze und deren Operationen bei einem Platz *Registry* registriert werden können. Danach kann man diese Operationen über eine URL ansprechen. Ist die URL eines Platzes bekannt, so kann ein spezieller Agent (*InteractiveRequestAgent*) zu dem Platz reisen und eine Operation aufrufen, die durch die URL spezifiziert wird.

Mit einem Gesuch (*Petition*) kann ein Treffen zwischen Agenten vermittelt werden. Besucht ein Agent, der die Mixin Klasse *MeetingAgent* besitzt, einen Platz, der die Mixin Klasse *MeetingPlace* besitzt, so kann der Agent andere Agenten, die ebenfalls die Mixin Klasse *MeetingAgent* besitzen, treffen. Um einen anderen Agenten treffen zu können, muß der Agent die ***meet*** Operation des Platzes mit einem Gesuch aufrufen. Ein Gesuch kann aus einem *telename* und Namen einer Klasse des zu treffenden Agenten bestehen. Gibt es einen Agenten, der das Gesuch erfüllt, so wird von der Engine seine ***meeting*** Operation aufgerufen und er erfährt den *telename*, den Namen der Klasse und das Gesuch des Agenten, der einen anderen Agenten treffen möchte. Durch das Auslösen einer Exception *MeetingDenied* kann der Agent das Gesuch ablehnen, aber er erhält keine Referenz auf den Agenten, welcher ihn treffen möchte (wie bei ***entering***, wenn ein Agent versucht, einen Platz zu besuchen). Kommt ein Treffen zustande, so besitzen beide Agenten eine Referenz auf jeweils den anderen und können durch objektorientierte Techniken Dienste des anderen in Anspruch nehmen.

3.8 Plätze und Agenten

Das Herzstück jeder Telescriptanwendung sind Plätze und Agenten. Es gibt verschiedenen Gründe, wann ein Platz oder ein Agent als Anwendung eingesetzt werden soll.

- Ein Agent wird benutzt, wenn eine Anwendung mit mehr als einem Platz interagieren soll, da Plätze nicht reisen können.
- Ist eine Anwendung stationär und soll eine GUI besitzen, so muß sie durch einen Platz realisiert werden.
- Eine Anwendung, die ständig verbessert und erweitert wird, sollte als Agent realisiert werden, da erst die alte Version der Anwendung in einer Engine zerstört werden muß, bevor die neue Version installiert werden kann. (Es können nie zwei verschiedene Klassendefinitionen mit dem selben Namen in einer Engine existieren.) Dies hat zur Folge, daß alle Agenten in diesem Platz zerstört werden. (Die maximale Lebenszeit eines Agenten könnte von einem Platz begrenzt werden, was aber eine starke Einschränkung ist.) Bei der Zerstörung eines Agenten werden nur die Verbindungen zu anderen Prozessen abgebrochen.

3.9 Fehlerbehandlung

Ein Fehler, der bei der Ausführung eines Telescriptprogrammes auftritt, löst eine Exception aus. Diese kann abgefangen werden. Wird sie nicht abgefangen, wird sie an den Aufrufer der Operation weitergereicht, usw. Wird sie nicht abgefangen, führt dies zur Beendigung des Prozesses, d.h. ein Platz oder Agent terminiert.

Da die Sprache Telescript persistent ist, überleben alle Objekte in einer Engine einen Engineabsturz z.B. durch Absturz des Computers. Ein Agent überlebt auch einen Absturz eines Platzes, den er gerade besucht, sofern dieser sofort wieder automatisch gestartet werden konnte. Diese Eigenschaft erhöht den Abstraktionsgrad eines Agenten, reduziert seine Größe und die Zuverlässigkeit des Agentensystems steigt.

3.10 Nachteile

- Es gibt keine Informationen, wie die Sprache implementiert wird.
- Es gibt noch viele Sicherheitseinschränkungen:
  - Es gibt ein weites Spektrum von Sicherheitstechniken, aber es wird dem Entwickler überlassen, diese zu nutzen. Standardmäßig werden nur wenige Sicherheitstechniken eingesetzt.
  - Eine aufgerufene Operation kann unerwünschte Seiteneffekte besitzen, da diese Operation immer Zugriff auf globale Variablen z.B. *process*, *owner*, usw. besitzt.
  - Die Permits, die ein Agent in einem Platz, zu dem er reist, besitzen möchte, werden ihm nicht garantiert. Der Platz hat das Recht, sie einzuschränken.
  - Jeder Prozeß kann alle Events versenden, auch die, die semantisch nur von der Engine versendet werden dürften.
  - Ein Platz kann Agenten, die seinen Platz betreten möchten, kontrollieren, aber nicht die Agenten, die einen seiner Subplätze betreten möchten. Dies erhöht den Aufwand, da jeder Platz möglichst viele Kontrollen durchführen sollte.
- Es werden viele Konzepte angeboten. Wird eine flexible Anwendung benötigt, sollte sie möglichst alle Konzepte nutzen und unterstützen, wodurch die Anwendung sehr umfangreich wird.

3.11 Zukunft

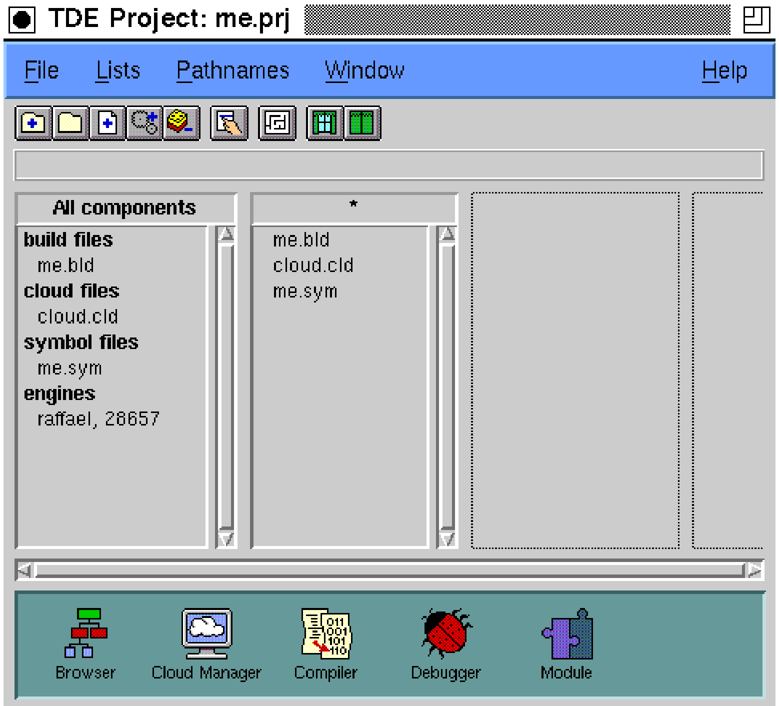
- Die Sicherheit von Kommunikationsverbindung und die Authentifizierung von Engines soll verbessert werden.
- Es soll globale Kommunikation eingeführt werden.
- Es soll eine API für den Zugriff auf Datenbanken in Tabriz AgentWare integriert werden.
- In der nächsten Version werden einige Klassen der TDE-Klassenbibliothek verändert.

4 Umgang mit Tabriz AgentWare

Das getestete Telescript System „Tabriz AgentWare V1.0p2“ ist eine kommerzielle Version für den nichtkommerziellen Einsatz. Für den kommerziellen Einsatz gibt es spezielle Lizenzen.

In dieser Version wurde stärker auf Benutzerfreundlichkeit als auf Sicherheit wert gelegt. Es sind Versionen für HP 700/800, Sun SPARCstation und SGI IRIS Indiego inbegriffen. Es soll eine Version für Windows NT in Kürze veröffentlicht werden.

Die TDE ist voll über ein GUI steuerbar.



Es wird weiterhin nur ein Editor für das Erstellen von Programmen benötigt.

Wird eine Engine mit HTML–API benutzt, so können alle Engineeinstellungen über diese Schnittstelle verändert werden.

4.1 Probleme und Fehler

- Die Hardware– bzw. Softwarevoraussetzungen an den Computer bzw. das Betriebsystem sind sehr umfangreich. Ich besaß diese erst nach drei Monaten.
- Sehr viel mitgelieferte Literatur war ausschließlich elektronisch in HTML– und z.T. in PDF–Format. Das verkleinerte Ausdrucken war sehr umständlich und zeitaufwendig. (In den Dokumenten fehlten an vielen Stellen Links bzw. Links referenzierten auf Dateien, die nicht existierten.)
- Das Installationsskript war nicht für IRIX 6.2 ausgelegt und mußte angepaßt werden. Die Installationsanleitung beschrieb die Installation und daß die TDE automatisch nach der Installation gestartet wird, aber nicht wie die gestartete Engine und Webserver korrekt angehalten werden können.

- Ein Beispielprogramm fehlte und ein weiteres konnte nicht korrekt installiert werden.
- Die Internetverbindung zu General Magic war sehr schlecht, so daß die Übertragung der Patches (jeweils ca. 16Mbyte) lange Zeit fehlschlug.
- Der angegebene Plattenplatzbedarf von 200Mbyte reichte nicht für die Installation der Software, Tests und Entwicklung der Beispielanwendung. Es mußten alle PDF Dokumente und die Daten jedes abgeschlossenen Tests gelöscht werden.
- Die Onlinerecherche war durch häufig verstopfte Datenautobahnen mühsam. Beispielsweise konnte während drei Tagen eine 89kbyte große HTML-Seite von France Telecom nicht geladen werden. Außerdem gab es über Telescript Tausende unbrauchbarer Seiten, die eine effektive Recherche sehr erschwerten.
- Die Autorität und die Identität werden im Source Viewer des TDE vertauscht angezeigt.
- Aktuell: Seit Ende Januar 1997 sind über Tabriz auf den General Magic Webserver [GenMagic] keine Informationen mehr zu erhalten; der Server wird umgestaltet.

4.2 Hilfe und Tips

Für Telescriptentwickler gibt es eine Mailing-Liste:

`TSCRIPT@BROWNVM.BROWN.EDU`

Für Magic Cap gibt es eine Mailing-Liste und eine Newsgroup:

`MAGICDEV@BROWNVM.BROWN.EDU`   `comp.os.magic-cap`

Weitere Informationen erhält man auch vom General Magic Webserver [GenMagic] im Verzeichnis `Develop/`. Desweiteren werden alle mitgelieferten sowie zusätzliche Dokumente (in HTML– und PDF–Format) auf diesem Webserver angeboten.

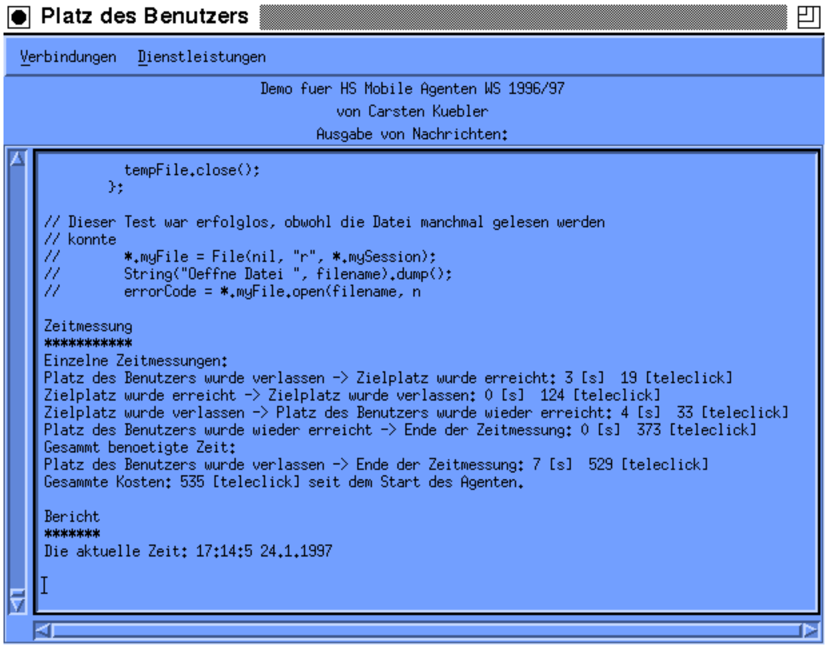
Alle im Lieferumfang enthaltenen Webbeispiele können auf dem Webserver getestet werden:

`http://freeagent.genmagic.com:3100/`

5 Beispielanwendung

Das Beispiel wurde folgendermaßen realisiert:

- Es wurde ein Platz des Benutzers mit GUI erstellt.



Durch Interaktion mit dem Platz konnte der Benutzer einen Dateisuchagenten mit *ticket* und *filename* versenden.

- Der Dateisuchagent ging zu dem mit *ticket* angegebenen Platz und versuchte, verschiedene Dateiserver zu besuchen und von ihnen die Datei mit *filename* zu erhalten. Danach versuchte der Dateisuchagent, zum Platz des Benutzers zurückzureisen.

- Der Dateiserver überprüfte die Autorität des Agenten, der eine Dateianfrage stellte. Agenten mit einer fremden Autorität, d.h. einer anderen als die des Dateiservers bzw. der Region, mußten sich erst registrieren lassen. Dies konnte ein Agent, indem er eine Anfrage nach einer existierenden Datei stellte. Bei der nächsten Dateianfrage eines Agenten (unter dieser Autorität) an den Dateiserver erhielt er die angefragte Datei.

5.1 Zeitmessungen

Da die Hardwarevoraussetzungen sehr umfangreich waren, konnten die Zeitmessungen nur auf einem Computer durchgeführt werden.

Es wurde ein lokaler Test mit einer Region und ein Test mit 2 Regionen durchgeführt. Beim Test mit 2 Regionen gab es drei Protokolle (UDP, UDP wireless, TCP), mit denen Agenten zwischen Engines ausgetauscht werden konnten.

Das Szenario bestand aus einem Platz des Benutzers und zwei Dateiservern, die Dateien aus verschiedenen Verzeichnissen laden konnten. Die übertragene Datei hatte eine Länge von 38kbyte.

Im Beispiel 1 hatte der zweite Dateiserver Zugriff auf die Datei und im Beispiel 2 der erste. Bei Tests mit zwei Regionen muß beachtet werden, daß beim ersten Mal die Verbindung von mir zwischen den Engines erstmalig benutzt wurde (Ausnahme Nr. 10) und der Agent von keinem Dateiserver eine Datei erhalten hat, da der erste Besuch zur Registrierung der Autorität des Agenten verwendet wurde. Die Authentifizierung entfiel bei der Verwendung einer Region, da alle Plätze dieselbe Autorität besaßen.

Da Telescript nur Zeitmessungen mit einer Genauigkeit von einer Sekunde anbietet, sind diese Aussagen nur schlecht vergleichbar und sehr ungenau. Es wurden jedoch noch die benötigten Teleclicks gemessen, die eine Aussage über die verbrauchte Rechenzeit geben, die auf Kosten des Agenten gingen. (Das Laden einer Datei ging nicht auf Kosten des Agenten, da der Dateizugriff unter der Autorität des Dateiservers realisiert wurde. Dies hat zur Folge, daß die Teleclicks innerhalb des Dateiservers bei jedem Beispiel „konstant“ (Ausnahme Nr. 10) waren.)

Gemessen wurde die Reisezeit vom Benutzerplatz zum Platz (der durch das *ticket* spezifiziert wurde), die Zeit innerhalb dieses Platzes (incl. aller besuchten Dateiserver) und die Zeit für die Rückreise des Agenten. Es wurde auch die Zeit für die Ausgabe eines Reports dieses Agenten und die insgesamt benötigte Zeit angegeben. Die Zeit wurde in [teleclick] bzw. [s] gemessen.

Nr.	Beispiel	Komm.art	Hinr.	D. Server	Rückr.	Ausg.	$\sum$ Zeit
1.	1 (1. mal)	lokal	14/ 0	123/ 1	31/ 0	373/1	521/ 2
2.	1 (2. mal)	lokal	14/ 0	123/ 0	31/ 0	373/1	521/ 1
3.	1 (3. mal)	lokal	14/ 0	123/ 0	31/ 0	373/0	521/ 0
4.	1 (4. mal)	lokal	14/ 0	123/ 1	31/ 0	373/0	521/ 1
5.	2 (1. mal)	lokal	14/ 0	88/ 0	31/ 0	381/1	494/ 1
6.	2 (2. mal)	lokal	14/ 0	88/ 0	31/ 0	381/0	494/ 0
7.	1 (1. mal)	UDP	19/15	125/ 1	33/ 5	298/0	455/21
8.	1 (2. mal)	UDP	19/ 3	124/ 0	33/ 4	373/0	529/ 7
9.	1 (3. mal)	UDP	19/ 3	124/ 0	33/ 3	373/1	529/ 7
10.	2 (1. mal)	UDP	19/ 4	124/ 0	33/ 3	298/0	454/ 8
11.	2 (2. mal)	UDP	19/ 3	88/ 0	33/ 4	381/0	501/ 7
12.	1 (1. mal)	UDP(w)	19/18	125/ 1	33/ 3	298/1	455/23
13.	1 (2. mal)	UDP(w)	19/ 8	124/ 0	33/ 4	373/0	529/12
14.	1 (1. mal)	TCP	19/13	125/ 0	33/ 3	298/0	455/16
15.	1 (2. mal)	TCP	19/ 3	124/ 0	33/ 2	373/1	529/ 6
16.	1 (3. mal)	TCP	19/ 3	124/ 0	33/ 5	373/0	529/ 8

5.2 Programm

Als Grundlage für das Beispielprogramm wurde das *multiengine* Beispiel von General Magic benutzt. Der *AbstractUserAgent* und der *FileServerPlace* sind vereinfachte und angepaßte Versionen des *StockWatcherAgent* und *StockMarketPlace*. Die Rechte am *multiengine* Beispiel besitzt General Magic, Inc.

5.2.1 Das Programm des Benutzerplatzes incl. Dateisuchagent

```
// Diese Demo wurde von Carsten K"ubler f"ur das
// Hauptseminar Mobile Agenten WS 1996/97 erstellt.

#include "tl.i"

ANY_SCRIPT_PRELUDE

UserInterface: module = (

<< "loading userPlace.t..." dump >>;
```

```

// Eine Quelldatei bindet seine eigene .d Datei ein und alle .i Dateien, die
// die Klasse in ihrer Implementierung und nicht in ihrem Interface benutzt

#include "userPlace.d"
```

... weitere Includes ...

```
// ***************************************************************************
// Eine Mixinklasse, die den Platz als UserPlatz auszeichnet
// ***************************************************************************


  HasUserPlaceAPI: mixin class (Place) = (

    public
      // Es darf immer nur ein berechtigter Proze"s im Ausgabefenster Text
      // ausgaben, damit die Ausgabe atomar ist. Ein fremder Proze"s
      // darf die Ressource nicht lesen.
      outputResource: readonly Resource with (
        get: op() Resource = {
          // Nicht jeder Proze"s darf Ausgaben erzeugen!!
          if (process.name.authority != *.name.authority) &&
             (process.name.authority != *.address.authority) {
            throw AccessRestriction('getOutput);
          };
          return privateOutputResource;
        };
      );

      // Die Liste der bekannten Engines mit Tickets darf nur exklusiv
      // ge"andert, aber gemeinsam gelesen werden. Ein fremder Proze"s
      // darf die Ressource nicht lesen.
      engineResource: readonly Resource with (
        get: op() Resource = {
          if (process.name.authority != *.name.authority) &&
             (process.name.authority != *.address.authority) {
            throw AccessRestriction('getEngine);
          };
          return privateEngineResource;
        };
      );


      // Die Liste der bekannten Engines mit Tickets. Ein fremder Proze"s
      // darf die Liste nicht lesen.
      knownEngines: readonly DictOwned[String, Ticket, Equal] with (
        get: op() DictOwned[String, Ticket, Equal] = {
          if (process.name.authority != *.name.authority) &&
             (process.name.authority != *.address.authority) {
            throw AccessRestriction('getKnownEngines);
          };
          return privateKnownEngines;
        };
      );


    private
      // Die Ressourcen und die Liste der Engines mit Ticket
      privateOutputResource: Resource;
      privateEngineResource: Resource;
      privateKnownEngines:   DictOwned[String, Ticket, Equal];


    public
      initialize: op() = {
        ^;
```

```

        privateOutputResource = Resource();
        privateEngineResource = Resource();
        privateKnownEngines = DictOwned[String, Ticket, Equal]();
      };


      // Eine "Uberschrift und eine Erkl"arung sollen ausgegeben werden.
      displayText: abstract op(shortTitle:     String | Nil;
                               theExplanation: String | Nil);


      // Eine "Uberschrift soll ausgegeben werden und der Benutzer soll einen
      // String eingeben k"onnen. Im Fehlerfall ist nil zur"uckzugeben.
      getString: abstract op(shortTitle: String | Nil) owned String | Nil;


      // Eine Exeption soll ausgegeben werden.
      reportError: abstract op(theException: Exception);


      // Ein kurzer Bericht soll erscheinen.
      report: abstract op(theExplanation: String);
  );



// ***************************************************************************
// Eine Mixinklasse, die den Platz als Fileserver auszeichnet
// ***************************************************************************


  HasFileServerAPI: mixin class (Place) = (
```

... siehe Dateiserverprogramm ...

```
  );



// ***************************************************************************
// Der Platz eines Benutzers
// ***************************************************************************


  UserPlace: class (Place, EventProcess, TDKPublished, HasUserPlaceAPI,
                    TDKWidgetEnabled, GUI) = (

    private
      myForm:               Form | Nil;
      myMenuBar:            MenuBar | Nil;
      myMenus:              List[Menu,Equal] | Nil;
      myOutputArea:         OutputArea | Nil;
      myStringInputDialog:  StringInputDialog | Nil;
      mySelectEngineDialog: SelectEngineDialog | Nil;
      myChooseEngineDialog: ChooseTicketToEngineDialog | Nil;


    public
      initialize: op(/* parent shell: */       Shell | Nil;
                     /* parent GUI session: */ TDKSession | Nil;
                     /* host: */               String | Nil;
                     /* port: */               Integer | Nil;
                     /* listen port: */        Integer | Nil;
                                               Permit;
                                               Permit | Nil;
                                               Integer | Nil;
```

```
                                             OctetString | Nil) = {
      ^;
      // Keine Events empfangen.
      *.disableEvents(nil);
    };


    live: sponsored op(Exception | Nil) = {
      // Mu"s hier geschehen, da man waehrend der Initialisierung
      // mit here keine Referenz auf einen Platz bekommt.
      *.addEngineToList(Pair[String, Ticket](String("Sweet Home"),
                                             Ticket(here.name.copy(),
                                                    here.address.copy(),
                                                    here.class.name.copy())));
      // Die live Operation der Superklasse aufrufen
      ^;
    };


    // Diese Operation wird beim Start von einer Superklasse aufgerufen.
    setUpDisplay: op() = {
```

... das realisierte GUI ...

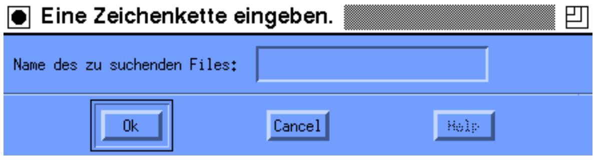

```
    };


  // Alle Methoden, die von einem Men"ue aufgerufen werden m"ussen "public"
  // sein, sonst k"onnen sie nicht aufgerufen werden.
  public

    // Eine einfache Eingabem"oglichkeit einer neuen Engine in einem Widget.
    addEngine: op() = {
      addressTicket: Pair[String, Ticket] | Nil;

      own {
        use *.shellAvailable when Set[Identifier, Equal]('Available) {
          // Widget erzeugen, wenn es noch nicht existiert.
          if mySelectEngineDialog == nil {
            mySelectEngineDialog = SelectEngineDialog(*.myShell);
          };

          // Widget sichtbar und nach der Eingabe wieder unsichtbar machen.
          addressTicket = mySelectEngineDialog.prompt();
          if addressTicket == nil {
            *.report("Keine neue Engine angebeben!");
```

```
        } else {
          *.addEngineToList(addressTicket);
        };
      } after 600 {
        throw Busy();
      };
    };
   };


  // Den Platz beenden. In der live Operation einer Superkasse ist eine
  // Mainloop eingebaut.
  doQuit: op() = {
    *.mySession@WidgetMgr.myContinue = false;
  };


  // Mit dieser Operation wird ein Agent erzeugt und versendet.
  sendAgent: op() = {
    fileName: String | Nil;
    ticket:   Ticket | Nil;

    // Frage nach einem Namen einer Datei, die von einem Datei-Server
    // geholt werden soll.
    fileName = *.getString("Name des zu suchenden Files: ");
    if (fileName == nil) || (fileName.length == 0) {
      *.report("Es wurde kein Dateiname angegeben.");
      return;
    };

    // Lasse den Benutzer ein Ticket ausw"ahlen.
    ticket = *.getEngine();
    if (ticket == nil) {
      *.report("Es wurde keine Zielengine angegeben.");
      return;
    };

    *.report(String("Versende Agent um Datei '", fileName, "' zu bekommen."));

    // Agent erzeugen und an einen Platz schicken. Platz wird durch Ticket
    // festgelegt.
    FileSearchUserAgent(fileName, ticket, Permit());
  };


  // Gebe die aktuelle Uhrzeit aus.
  aktuelleZeit: op() = {
    *.report(MyTime("Die aktuelle Zeit").asString());
  };


// Alle abstrakten Operationen der Klasse HasUserPlaceAPI definieren.
public

  // Eine einfache Ausgabe zweier Strings
  displayText: op(shortTitle:     String | Nil;
                  theExplanation: String | Nil) = {
    own {
      use *.shellAvailable when Set[Identifier, Equal]('Available) {
        // Die Ausgabe geschieht auf dreimal, damit die Benutzung der
        // Ressource auch einen Sinn hat.
        use *.outputResource {
          output: String;

          if shortTitle != nil {
            output = String(shortTitle);
            output.append("\n");
            for i: Integer to shortTitle.length {
```

```
          output.append("*");
        };
        output.append("\n");
        myOutputArea.write(output);
      };
      if theExplanation != nil {
        output = String(theExplanation);
        output.append("\n");
        myOutputArea.write(output);
      };
      myOutputArea.write("\n");
    } after 600 {
      // Exkludivzugriff auf "*.outputResource" nicht innerhalb der
      // angegebenen Zeit erhalten.
      throw Busy();
    };
    *.flushToDisplay();
  } after 600 {
    throw Busy();
  };
 };
};


// Eine einfache Eingabem"oglichkeit in einem Widget.
getString: op(shortTitle: String | Nil) owned String | Nil = {
  own {
    use *.shellAvailable when Set[Identifier, Equal]('Available) {
      // Widget erzeugen, wenn es noch nicht existiert.
      if myStringInputDialog == nil {
        myStringInputDialog = StringInputDialog(*.myShell);
      };

      // Widget sichtbar und nach der Eingabe wieder unsichtbar machen.
      return myStringInputDialog.prompt(shortTitle);
    } after 600 {
      throw Busy();
    };
  };
};


// Eine einfache Ausgabem"oglichkeit f"ur Exeptions.
reportError: op(theException: Exception) = {
  *.displayText("Fehlermeldung", theException.dump2Chars());
};


// Eine einfache Ausgabem"oglichkeit f"ur einen Bericht.
report: op(theExplanation: String) = {
  *.displayText("Bericht", theExplanation);
};


// Private Operationen dieses Platzes
private
  // Erg"anze ein Ticket zu einer anderen Engine.
  addEngineToList: op(addressTicket: owned Pair[String, Ticket]) = {
    // Exklusiv einen Eintrag in die Liste der Engines mit Tickets machen.
    use place@HasUserPlaceAPI.engineResource {
      try {
        *.knownEngines.addOwned(addressTicket.object1, addressTicket.object2);
      } catch KeyInvalid {
      };
    } after 600 {
      throw Busy();
    };
  };
```

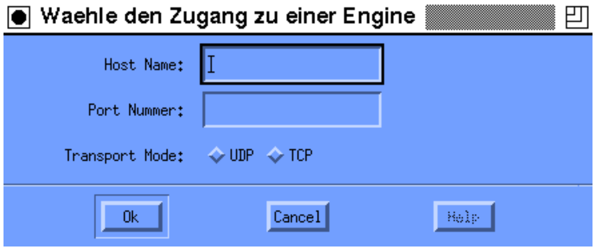
```


      // Der Benutzer kann eine Engine ausw"ahlen, zu der ein Ticket bekannt
      // ist und gebe das Ticket zur"uck.
      getEngine: op() owned Ticket | Nil = {
        own {
          use *.shellAvailable when Set[Identifier, Equal]('Available) {
            // Widget erzeugen, wenn es noch nicht existiert.
            if myChooseEngineDialog == nil {
              myChooseEngineDialog = ChooseTicketToEngineDialog(*.myShell);
            };

            // Widget sichtbar und nach der Eingabe wieder unsichtbar machen.
            return myChooseEngineDialog.prompt();
          } after 600 {
            throw Busy();
          };
        };
       };
  );



// ***************************************************************************
// Erzeugt ein Eingabewidget f"ur ein String
// ***************************************************************************


  StringInputDialog: class(CustomDialog, Sync) = (
    public
    prompt: op(shortTitle: String | Nil) owned String | Nil = {
    };
```

... das realisierte Widget ...



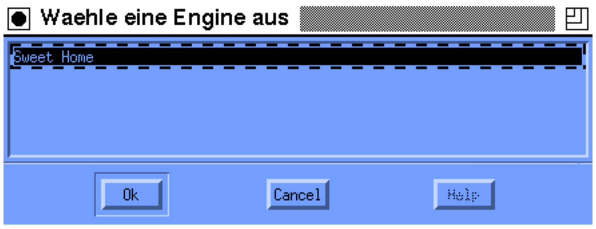

```
  );



// ***************************************************************************
// Erzeugt ein Eingabewidget f"ur eine neue Engine
// ***************************************************************************


  SelectEngineDialog: class(CustomDialog, Sync) = (
    public
      prompt: op() owned Pair[String, Ticket] | Nil = {
        return Pair[String, Ticket](String("Host ", theHost,
                                          "/Port ", thePort.asString()),
                                   theTicket);
      };
```

... das realisierte Widget ...




```
  );



// ***************************************************************************
// Erzeugt ein Eingabewidget f"ur die Auswahl einer Engine
// ***************************************************************************


  ChooseTicketToEngineDialog: class(CustomDialog, Sync) = (
    public
      prompt: op() owned Ticket | Nil = {
      };
```

... das realisierte Widget ...




```
  );



// ***************************************************************************
// Eine konkrete Realisierung eines Agenten, der in der Zielregion eine Datei
// suchen soll und diese am Ausgangsplatz wieder ausgeben soll
// Der Agent setzt folgendes Voraus:
// - Der Ausgangsplatz besitzt die Mixinklasse HasUserPlaceAPI
// - Die Datei-Server haben die Bezeichnung, die die Konstante FILE_SERVICE
//   beinhaltet
// - Der Datei-Server besitzt die Mixinklasse HasFileServerAPI
// ***************************************************************************


  FileSearchUserAgent: class (AbstractUserAgent) = (

    private
      leaveUsPlTime:    MyTime;
      arriveDestPlTime: MyTime;
      leaveDestPlTime:  MyTime;
      arriveUsPlTime:   MyTime;
      filename:         String;
      filedata:         String | Nil;


    public instance
      initialize: op(filename: copied String;
```

```
                             Ticket;
                             Permit;
                             Permit | Nil;
                             Integer | Nil) = {
      ^;

      filedata = nil;
    };


  private
    // Diese Variablen sollten in der aktuellen Engineversion nicht lokal
    // sein, da beim Verlassen einer Engine zweifelhaft ist, ob ihre Daten
    // auf der Reise nicht verloren gehen.
    marketsToTry: List[Ticket,Equal] | Nil;
    serverCount:  Integer;
    restricts:    Permit;


    // Alle Vorbereitungen vor der Reise des Agenten zu Zielplatz.
    prepareToGo: op() = {
      // Die Klasse MyTime auf die Reise mitnehmen, damit der Agent sie beim
      // Zielplatz benutzen kann.
      *.publicPackages.include(Package[Object, Object, Equal]
                               (nil, nil,
                                'MyTime, MyTime));
      leaveUsPlTime = MyTime("Platz des Benutzers wurde verlassen");
    };


    // Alles am Zielplatz erledigen, d.h. Versuche eine Datei zu bekommen.
    carryOutPurpose: op() = {
      cardFound:     AddressCard | Nil;
      serverFacility: Facility | Nil;

      arriveDestPlTime = MyTime("Zielplatz wurde erreicht");
      serverCount = 0; // Ein Z"ahler, wieviel Datei Server besucht wurden.

      // Hole die Adresskarte der Region und schaue ob es einen Datei-Server
      // gibt.
      cardFound = *.getPublishedObject(REGION_CARD_NAME, nil)@AddressCard;
      if cardFound != nil {
        // Alle Pl"atze, die scheinbar den Dienst anbieten, anfordern.
        serverFacility = cardFound.getFacility(FILE_SERVICE);
      };
      if serverFacility != nil {
        // Es gibt eine Liste von Anbietern dieses Dienstes.
        marketsToTry = serverFacility.template@List[Ticket,Equal];
        cardFound = nil;
        serverFacility = nil;

        // Was verbiete ich allen Operationen, die ich bei einem Zielplatz
        // aufrufe.
        restricts = Permit();
        restricts.canCreate = false;
        restricts.canDeny = false;
        restricts.canGo = false;
        restricts.canGrant = false;
        restricts.canRestart = false;
        restricts.canSend = false;

        // Versuche alle Pl"atze zu besuchen, die diesen Dienst anbieten.
        for aTicket: Ticket in marketsToTry {
          serverCount = serverCount + 1; // Besuche einen Platz.
          try {
            // Versuche zu dem Dienst zu kommen und seinen Dienst in
            // Anspruch zu nehmen.
            // Reise ...
```

```
        *.go(aTicket);
        // ... Reise
        own {
          // Die Daten vom Datei-Server holen. Sie werden als Kopie
          // zur"uckgegeben und Kopien geh"oren dem Agent, da die Kopie
          // in einer own Umgebung durchgef"uhrt wird.
          // Die Einschr"ankungen an den aktuellen Zustand anpassen.
          restricts.age = *.age + 60; // Die max. Dauer der Operation.
          restricts.charges = *.charges + 10000; // Max. Teleclicks.

          try {
            // Schr"anke die Permits stark ein, so wie restricts angibt.
            restrict restricts {
              // Wer wei"s was der Dateiserver alles macht, da seine
              // Operation nicht gesponsert ist, geht alles auf die
              // Kosten des Agenten.
              filedata = place@HasFileServerAPI.readFile(filename);
            };
          } catch PermitViolated {
            // Die Operation wollte gegen die Einschr"ankungen, die
            // der Agent vorgegeben hat, versto"sen.
          };
          if filedata != nil {
            // Datei wurde bei diesem Server gefunden. Es m"ussen keine
            // weiteren Dateiserver besucht werden.
            break; // for aTicket: Ticket in marketsToTry
          };
        };
      } catch theError: Exception {
        // Es ist egal was passiert ist. An diesem Platz hat der Agent
        // die Datei nicht erhalten! Versuche es einfach bei einem
        // anderen.
      };
    };
  } else {
    // In dieser Region gibt es keinen Datei Server. Aufgabe erledigt!
  };

  leaveDestPlTime = MyTime("Zielplatz wurde verlassen");
};


// Jetzt sollte der Agent wieder zu Hause sein, sofern nichts schief
// lief.
reportBack: op() = {
  arriveUsPlTime = MyTime("Platz des Benutzers wurde wieder erreicht");

  // Versuche bei diesem Platz die Ausgabe mit HasUserPlaceAPI.
  // Im Fegefeuer Platz gibt es die Mixinklasse nicht!
  try {
    timeMeasure: String;

    // Ausgabe erzeugen.
    place@HasUserPlaceAPI.report(String("Agent ist von der Reise wieder",
                                         " da.\nEr hat ",
                                        serverCount.asString(),
                                        " Datei Server besucht."));
    if filedata != nil {
      // Datei wurde erhalten und wird jetzt ausgegeben.
      place@HasUserPlaceAPI.displayText(String("Inhalt der Datei: ",
                                               filename), filedata);
    }
    else if *.failureCause != nil {
      // Eine Exeption wurde abgefangen und wird ausgegeben.
      place@HasUserPlaceAPI.reportError(*.failureCause);
    }
    else {
      // Keine Datei erhalten und keine Exeption aufgetreten.
```

```
            place@HasUserPlaceAPI.report(String("Datei '", filename,
                                                "' nirgendwo erhalten."));
          };
          // Gib die gemessenen Zeiten aus. Sekunden sind zu grob, Teleclicks
          // sind feiner, es wird aber nur gemessen, wieviel Rechenzeit unter
          // den Permits des Agenten ben"otigt wurde. Die Einheit von
          // Teleclicks ist nirgends angegeben.
          timeMeasure =
            String("Einzelne Zeitmessungen:\n",
                   arriveDestPlTime.intervalAsString(leaveUsPlTime),   "\n",
                   leaveDestPlTime.intervalAsString(arriveDestPlTime), "\n",
                   arriveUsPlTime.intervalAsString(leaveDestPlTime),   "\n",
                   MyTime("Ende der Zeitmessung").intervalAsString(arriveUsPlTime),
                   "\nGesammt benoetigte Zeit:\n",
                   MyTime("Ende der Zeitmessung").intervalAsString(leaveUsPlTime),
                   "\nGesammte Kosten: ", *.charges.asString(),
                   " [teleclick] seit dem Start des Agenten.");
          place@HasUserPlaceAPI.displayText("Zeitmessung", timeMeasure);
        } catch ex: Exception {
          // Platz hat nicht die Mixinklasse HasUserPlaceAPI. Gebe trotzdem
          // eine Fehlermeldung aus. Diese Erscheint mindestens im Engine-
         // fenster, in der sich der Agent gerade aufh"alt.
          String("Ausgabe des Dateisuchagenten fehlgeschlagen: ",
                 ex.dump2Chars()).dump();
          return;
        };
      };
  );



// ****************************************************************************
// Eine abstrakte Klasse eines Agenten des Benutzers
// ****************************************************************************


  AbstractUserAgent: abstract class (Agent, TDKGetPublished) = (
    private
      destinationTicket: Ticket;
      returnTicket:      Ticket;
      failureCause:      Exception | Nil;

    public instance
      // W"ahrend der Initialisierung wird eine Kopie des Tickets zum Ziel
      // erstellt um ein Ticket zur"uck zum Ausgangsplatz zu erzeugen und
      // ein Attribut wird mit "nil" initialisiert, das eine Exeption
      // enth"alt, wenn ein Fehler auftrat.
      initialize: op(destinationTicket: copied Ticket;
                                        Permit;
                                        Permit | Nil;
                                        Integer | Nil) = {
        ^;

        returnTicket = *.makeReturnTicket(destinationTicket);
        failureCause = nil;
      };


      // Erledigt das Reisen vom Ausgangsplatz zum Zielplatz und wieder
      // zur"uck.
      live: sponsored op(Exception | Nil) = {
        returnStub: TicketStub | Nil;

        returnStub = nil;
        try {
          // Erledige alle Vorbereitungen vor der Reise zum Zielplatz.
         *.prepareToGo();

```

```
          // Reise ...
          returnStub = *.go(destinationTicket);
          // ... Reise beendet

          // Wenn man am Ziel angelangt ist, dann f"uhre dort Aktionen durch.
          *.carryOutPurpose();
        } catch outboundError:  Exception {
          // Ein Fehler ist bei der Vorbereitung, der Reise oder bei den
          // Aktionen am Ziel aufgetreten. Merke die Exeption f"ur den
          // Bericht beim Ausgangsplatz
          failureCause = outboundError;
        };

        // Es wurde alles erledigt oder ein Fehler abgefangen. Versuche wieder
        // zur"uck zu kommen und wenn das nicht m"oglich ist, lasse eine
        // Fehlermeldung mit MagicBackstop loggen.
        try {
          if (returnStub != nil) && (returnTicket.wayOut == nil) {
            returnTicket.wayOut = returnStub.wayBack;
          };

          // Reise ...
          *.go(returnTicket);
          // ... Reise beendet

          // Werte alles aus und mache einen Bericht.
          *.reportBack();
        } catch returnError: Exception {
```

... auf jeden Fall eine Fehlermeldung festhalten ...

```
        };
      };


    // Alle Operationen, die vor, w"ahrend, nach der Reise von diesem
    // abstrakten Agenten aufgerufen werden.
    private
      // Alle Vorbereitungen eines Agenten, bevor er geht
      prepareToGo: op() = {};


      // Alle Aktivit"aten am Zielplatz
      carryOutPurpose: abstract op();


      // Die Auswertung der Ereignisse am Ausgangsplatz
      reportBack: abstract op();


    private
      // Erstelle aus dem Ticket zu Zielplatz ein Ticket f"ur die R"uckreise
      makeReturnTicket: op(theDestination: Ticket) Ticket = {
        ticketBack:    Ticket;
        outboundMeans: Means | Nil;
        returnMeans:   Means | Nil;
        optGetter :=   ConfigOptionGetter();

        ticketBack = Ticket(Telename(place.name.authority.copy()),
                            place.address.copy(),
```

```
                             place.class.name.copy());

        // Wenn der Agent "uber eine LAN oder TCP Verbindung geht, mu"s das
        // Ticket noch mit der Adresse und dem Port erg"anzen.
        outboundMeans = theDestination.wayOut?.means;
        returnMeans = nil;
        if outboundMeans is LANMeans {
          returnMeans = LANMeans(optGetter.getOption(ENGINE_IP_OPTION,
                                                    DFT_ENGINE_IP)@String,
                                 optGetter.getOption(ENGINE_PORT_OPTION,
                                                    DFT_ENGINE_PORT)@Integer);
        } else if outboundMeans is TCPMeans {
          returnMeans = TCPMeans(optGetter.getOption(ENGINE_IP_OPTION,
                                                    DFT_ENGINE_IP)@String,
                                 optGetter.getOption(ENGINE_TCP_PORT_OPTION,
                                                    DFT_ENGINE_PORT)@Integer);
        };
        if returnMeans != nil {
          ticketBack.wayOut = Way(nil, returnMeans);
        };
        return ticketBack;
      };
  );



// ***************************************************************************
// Erstelle eine neue Klasse, die einen Kommentar zur aktuellen Zeit
// kennt und einfache Strings aus den Zeiten machen kann. Es sollen auch die
// Teleclicks gemessen werden.
// ***************************************************************************


  MyTime: class(Time) = (

    public
      timeName:   String;
      teleClicks: Integer;


      // Initialisierung beider Attribute! 1. durch Parameter
      //                                   2. durch Zuweisung
      initialize: op(timeName: String) = {
        ^;
        teleClicks = process.charges;
      };


      // Bildet einen String aus dem Namen des Zeitpunktes und der Zeit.
      asString: op() String = {
        calendarTime: CalendarTime;
        string:       String;

        calendarTime = *.localize();
```

```
        string = String(*.timeName, ": ",
                        calendarTime.hour.asString(), ":",
                        calendarTime.minute.asString(), ":",
                        calendarTime.second.asString(), " ",
                        calendarTime.day.asString(), ".",
                        calendarTime.month.asString(), ".",
                        calendarTime.year.asString());
      };


      // Bildet einen String aus den Namen der Zeitpunkte und der
      // Intervallzeit. Gebe auch die Teleclickdifferenz aus. Hier k"onnte
      // man noch testen, ob die Teleclicks vom gleichen Proze"s sind und
      // nur dann eine Differenz bilden.
      intervalAsString: op(subtrahend: MyTime) String = {
        interval:   Integer;
        intervalTC: Integer;
        string:     String;

        interval = *.interval(subtrahend);
        intervalTC = *.teleClicks - subtrahend.teleClicks;
        string = String(subtrahend.timeName, " -> ", *.timeName, ": ",
                        interval.asString(), " [s]  ",
                        intervalTC.asString(), " [teleclick]");
      };
  );



// ****************************************************************************
// Der Installer des UserPlaces. Er startet den Platz mit einer GUI.
// ****************************************************************************


  UserPlaceInstaller: class(Agent) = (

    private
      clientLocation:  String;


    public
      // Initialisiere Attribute mit Kopien ...
      initialize: op(clientLocation: copied String;
                                     Permit;
                                     Permit | Nil;
                                     Integer | Nil) = {
        ^;
      };


      live: sponsored op(Exception | Nil) = {
        optGetter:=  ConfigOptionGetter();

        // Erzeuge den Platz f"ur den User
```

```
        // Die "ubergebenen Werte werden f"ur das GUI ben"otigt
        UserPlace(nil, nil,
                  optGetter.getOption(CONSOLE_HOST_OPTION,
                                      DFT_CONSOLE_HOST)@String,
                  optGetter.getOption(CONSOLE_PORT_OPTION,
                                      DFT_CONSOLE_PORT)@Integer,
                  optGetter.getOption(CONSOLE_LPORT_OPTION,
                                      DFT_LISTEN_PORT)@Integer,
                  Permit(), nil, nil, clientLocation.asOctetString());
      };
  );

) // Das Ende des Moduls.

ANY_SCRIPT_CONCLUSION;
```

5.2.2 Das Programm des Dateiserverplatzes

```
// Diese Demo wurde von Carsten K"ubler f"ur das
// Hauptseminar Mobile Agenten WS 1996/97 erstellt.

#include "tl.i"

ANY_SCRIPT_PRELUDE

ServerInterface: module = (

<< "loading serverPlace.t..." dump >>;

// Eine Quelldatei bindet seine eigene .d Datei ein und alle .i Dateien, die
// die Klasse in ihrer Implementierung und nicht in ihrem Interface benutzt

#include "serverPlace.d"
```

... weitere Includes ...

```
// ***************************************************************************
// Eine Mixinklasse, die den Platz als Fileserver auszeichnet
// ***************************************************************************


  HasFileServerAPI: mixin class (Place) = (

    public

      initialize: op() = {
        ^;
      };


      // Dieser Operation wird ein Dateiname "ubergeben und eine Kopie
      // eines Strings der angeforderten Datei wird zur"uckgegeben.
      // Im Fehlerfall ist nil zur"uckzugeben.
      readFile: abstract op(filename: String) copied String | Nil;
  );
```

```
// ***************************************************************************
// Eine Exception, die der FileAccessServer ausl"osen kann, wenn ein Agent
// unerlaubt einen Dienst anfordert.
// ***************************************************************************


 FileServerAccessViolation: class (Exception) = ();



// ***************************************************************************
// Erstelle einen FileAccessServer
// ***************************************************************************


 FileServerPlace: class (Place, MeetingPlace, TSEventEnabled, TDKPublished,
                         HasFileServerAPI) = (

   public
     // Der Datei Server macht seine Adresskarte "uber ein Attribut lesbar.
     // Sie kann nicht "uberschrieben werden und es kann auch nicht ihre Werte
     // ver"andert werden. Die Anmerkung copied hat den Effekt, da"s dem
     // Aufrufer eine Kopie zur"uckgegeben wird, die ihm geh"ort.
     serverPlaceCard: readonly copied AddressCard;


   private
     // Der Pfad, den der Datei Server vor jeden Dateinamen anh"angt. Diesen
     // braucht nicht jeder zu kennen, deshalb ist er private.
     serverPath:        String;
     // Eine Liste von Autorit"aten, die diesen Dienste in Anspruch nehmen d"urfen.
     allowedAuthoritys: List[OctetString, Equal];
     // Es darf immer nur einer Eintr"age in die Liste von Autorit"aten machen.
     privateAuthorityResource: Resource;


   public instance
     // Die Attribute werden kopiert, damit sie nicht garbage collected
     // werden, sobald der FileServerInstaller stirbt, da ihm die Orginale
     // geh"oren.
     initialize: op(serverName: copied String;
                    serverPath: copied String;
                                Permit;
                                Permit | Nil;
                                Integer|Nil;
                                OctetString | Nil) = {
       thisCard:  ProviderAddressCard;

       ^;

       // Die Ressource initialisieren
```

```
    privateAuthorityResource = Resource();
    // Es wird eine Adre"skarte f"ur diesen Dateiserver erzeut.
    thisCard = ProviderAddressCard(
                 MagicmailName(serverName, List[NameKey, Equal]
                                (NameKey(SORT_KEY, 1, serverName.length))),
                 Telename(*.name.authority.copy()),
                 BUSINESS,
                 List[AddressLabel, Equal](ProviderLabel(*.address, nil, nil)));
    // Adre"skarte
    *.addFacilityLabel(thisCard);
    *.setAddressCard(thisCard);
    allowedAuthoritys = List[OctetString, Equal]();
  };


  // The public operation addFacilityLabel adds a facility for this file
  // server place to the given address card.  Its implementation assumes
  // that everyone else is obeying the convention that the FILE_SERVICE
  // facility holds a list of tickets (instead of just a single ticket).
  addFacilityLabel: op(theCard: AddressCard) = {
    fileServerFacility: Facility | Nil;
    hostIP:             String;
    hostPort:           Integer;
    optionGetter:       ConfigOptionGetter;

    // The facility's name says that this is (one of) the file servers
    // available on the engine.  The ticket points to this file server
    // place.
    // We can't use the operation getFacility on theCard because
    // getFacility returns the facility by copy.  We need to original
    // so that we can edit its list of tickets.
    fileServerFacility = nil;
    if (theCard.facilityListProperty != nil) {
      for aFacility: Facility in theCard.facilityList {
        if aFacility.facilityName == FILE_SERVICE {
          fileServerFacility = aFacility;
          break;
        };
      };
    };
    if fileServerFacility == nil {
      // Dies ist der erste Dateiserver unter dieser Autorit"at.
      fileServerFacility = Facility(FILE_SERVICE, List[Ticket,Equal]());
      if theCard.facilityList == nil {
        theCard.facilityList = List[Facility,Equal](fileServerFacility);
      } else {
        theCard.facilityList.include(fileServerFacility);
      };
    };
    fileServerFacility.template@List[Ticket,Equal].include(
      Ticket(Telename(*.name.authority.copy()),
             *.address.copy(), *.class.name.copy()));
  };

```

```

   live: sponsored op(Exception | Nil) = {
     // Alle Agenten unter der Autorit"at des Platzes und der Engine
     // haben sofortigen Zugriff auf die readFile Operation
     *.registerAuthority(*.name.authority);
     *.registerAuthority(*.address.authority);

     // Eine Endlosschleife! Die Agenten serben erst beim Ablauf ihrer
     // Permits und die m"ussen garnicht eingeschr"ankt sein.
     *.respondToAllEvents(); // In diesem Demo werden keine Events behandelt
   };


   // Unter welchen Bedingungen l"a"st der Platz den Agenten eintreten.
   // Die "ubergebenen Permits sind der Schnitt der eigenen (die bei der
   // Initialisierung des Agenten angegeben wurden), regionalen und
   // gew"unschten Permits.
   entering: sponsored op(who:      protected Telename;
                          theClass: protected ClassName!;
                          permit:   unprotected Permit;
                          ticket:   protected Ticket | Nil) Object = {
     try {
       // Allgemeine Einschr"ankungen aller Agenten, die diesen Platz
       // besuchen wollen.
       permit.canCreate = false; // Agent darf keine neuen Agenten erzeugen.
       permit.canDeny = false;
       permit.canSend = false; // Agent darf Platz nur mit go verlassen
       permit.canRestart = false;
       permit.canCharge = false;
       permit.canGrant = false;
       permit.age = 120; // Agent darf 2 Min. im Platz leben und danach
                         // stirbt er.
       if ((permit.priority == nil) || (permit.priority > *.priority)) {
         // Der Agent bekommt keine h"ohere Priorit"at als der Platz selbst
         // hat.
         permit.priority = *.priority - 5;
       };
       // Hier k"onnte noch eine spezifische Kontrolle von who und theClass
       // in Verbindung mit den Permits durchgef"uhrt werden.
       // Es wird aber jeder Agent hereingelassen!
       return Nil;
     } catch Exception {
       // Der Platz konnte die M"achtigkeit des Agenten nicht einschr"anken.
       // Der Platz l"a"st den Agenten nicht eintreten.
       throw OccupancyDenied();
     };
   };


private
   setAddressCard: op(theAddressCard: copied AddressCard) = {
     // We can update the property but not the attribute "serverPlaceCard".
     // The property is what we get if we refer to serverPlaceCard with no
     // preliminaries. *.serverPlaceCard or self.serverPlaceCard would
```

```
      // refer to the attribute.
      serverPlaceCard = theAddressCard;

      // It is a General Magic convention for a service or subscriber to
      // "publish" its address card in the service or subscriber's
      // outermost place under the key OWNER_CARD_NAME.  Do so.
      self.setPublishedObject(OWNER_CARD_NAME, serverPlaceCard);
      self.publishAsPackage();
    };


  public
    readFile: op(filename: String) copied String | Nil = {
      theSponsor: OctetString;

      theSponsor = sponsor.name.authority;
      // Hier werden Sicherheitschecks durchgef"uhrt.
      if allowedAuthoritys.examine(theSponsor) == nil {
        // Registriert den Aufrufer, falls er einen korrekten Dateinamen
        // "ubergeben hat. -> Sehr einfaches Kriterium !!! (Keine Interaktion
        // mit einem Benutzer n"otig! -> einfache Demo.)
        *.registerAuthorityByTest(theSponsor, filename);
        throw FileServerAccessViolation();
      };

      return *.readFilePrivate(filename);
    };


  private
    registerAuthorityByTest: sponsored op(theSponsor: OctetString;
                                          filename:   String) = {
      tempFile: TestFile;

      // Teste, ob es die Datei gibt.
      try {
        tempFile = TestFile(String(serverPath, filename), "r");
        tempFile.close();
        // Die Datei gibt es. Autorit"at wird registriert und kann n"achstes
        // mal Daten lesen.
        own {
          *.registerAuthority(theSponsor.copy());
        };
      } catch theException : Exception {
        tempFile.close();
        // Die Datei gibt es nicht. Autorit"at wird nicht registriert.
      };
    };


    registerAuthority: op(theSponsor: owned OctetString) = {
      use privateAuthorityResource { // Exklusiven Zugriff
        allowedAuthoritys.include(theSponsor);
      } after 600 { throw Busy(); };
```

```
      };


      // Gesponserte Operation, damit der Dateizugriff unter der Autorith"at
      // und den Permits des Platzes ausgef"uhrt wird. Da ein Dateizugriff
      // unter einer fremden Autorit"at nicht m"oglich sein kann.
      readFilePrivate: sponsored op(filename: String) String | Nil = {
        tempFile: TestFile;
        filedata: String | Nil;

        filedata = nil;
        try {
          tempFile = TestFile(String(serverPath, filename), "r");
          filedata = tempFile.read(51200); // Sollte f"ur die Demo ausreichen.
          tempFile.close();
        } catch theException : Exception {
          filedata = nil;
          tempFile.close();
        };

        return filedata;
      };
  );



// ****************************************************************************
// Der Installer eines Datei Servers  startet den Platz und meldet ihn an.
// ****************************************************************************


  FileServerInstaller: class(Agent) = (
    private
      serverPath:       String;
      serverName:       String;
      serverLocation:   String;
      portConfigOption: Identifier;


    public
      initialize: op(serverName:     copied String;
                     serverPath:     copied String;
                     serverLocation: copied String;
                                     Permit;
                                     Permit | Nil;
                                     Integer | Nil) = {
        ^;
      };


      live: sponsored op(Exception | Nil) = {
        regionCard:      AddressCard;
        theServer:       FileServerPlace;
        thisPlaceMenu:   ThumperModuled;
```

```
        optionGetter := ConfigOptionGetter();

        theServer = FileServerPlace(serverName, serverPath,
                                    Permit(), nil, nil,
                                    serverLocation.asOctetString());

        // Erg"anze den Dienst in der Adresskarte der Engine.
        regionCard = place@TDKPublished.getPublishedObject(REGION_CARD_NAME)
                         @AddressCard.unlockedCopy();
        theServer.addFacilityLabel(regionCard);
        place@TDKPublished.setPublishedObject(REGION_CARD_NAME, regionCard);
        place@TDKPublished.publishAsPackage();
      };
  );



) // Das Ende des Moduls.

ANY_SCRIPT_CONCLUSION;
```

Abkürzungsverzeichnis

API: Application Program Interface

EAF: External Application Framework

GUI: Graphic User Interface

HTML: Hypertext Markup Language

Magic Cap: Magic Communicating application platform

NTT: Nippon Telegraph and Telephone Corporation

PIC: Personal Intelligent Communicator

TCP: Transmission Control Protocol

TDE: Telescript Developer Environment

UDP: User Datagram Protocol

URL: Uniform Resource Locator

Anhang

A Common Agent Platform

Jim White, General Magic, 11 March 1996
jim_white@genmagic.com
Submitted to the Joint *W3C/OMG Workshop on Distributed Objects and Mobile Code*

In the last five years General Magic has developed and deployed Telescript, a technology for mobile agents. This paper sketches a common agent platform intended to allow any number of such technologies to coexist and interoperate.

Agent

An agent is a program that a person or organization vests with its authority, that can run unattended for a long time (e.g., a week), and that can meet and interact with other agents. The person or organization is the agent's *authority*.

One can distinguish two kinds of agent. A *stationary agent* executes on one computer system. A mobile agent can execute on different computer systems at different times of its life.

Agent platform

Agents will be written in a variety of programming languages. This creates a need for a *common agent platform*, a language-independent means for agent interaction and mobile agent migration. A server's implementation of the platform might offer a full range of services; a PC's might offer a subset. An application protocol would interconnect the implementations of different systems so that their agents can interact and so that their mobile agents can migrate between systems.

The platform would offer agent services to its clients. These clients would include *language environments*, the virtual machines and libraries for particular programming languages. A language environment would offer some or all of the agent services to programs written in that language.

Agent services

The platform could provide services like these:

- *Program naming.* Assign names to agent programs so as to distinguish one from another.
- *Program authentication.* Authenticate the implementors of agent programs.
- *Program migration.* Transport agent programs between computers. The name of a program might be used to detect that it was already present at its destination.
- *Agent naming.* Assign names to agents so as to distinguish one from another.
- *Agent authentication.* Authenticate the authorities of agents.
- *Agent migration.* Transport agents between computers. The mobile agent's destination might be specified indirectly as the computer that hosts a specified stationary agent.
- *Agent permissions.* Assign names to permissions, enforce an agent's assigned permissions, and let one agent grant permissions to other agents or re-negotiate its own.
- *Agent relationships.* Assign names to relationships, let agents begin and end relationships, and enforce rules of conduct for relationships.
- *Agent creation.* Let agents create other agents locally and remotely. A new agent might have the authority of the existing agent and either the same permissions or a subset of them.
- *Agent termination.* Terminate agents in an orderly fashion, e.g., by notifying other agents with which they have relationships.
- *Agent waiting.* Write to disk agents that must wait for long periods of time for events to occur (e.g., for relationships to be established).
- *Agent checkpointing.* Checkpoint agents to disk so that they survive crashes of their host systems. A transaction service might coordinate the checkpointing of related agents.
- *Agent interaction.* Let related agents interact. The means of interaction (e.g., events, pipes, and RPC) might depend upon whether the agents occupy the same or different systems.
- *Agent management.* Let network management and debugging agents monitor other agents. A debugging agent, e.g., might be able to suspend and resume agents under development.

Literatur

[AWDevLib96] *Tabriz AgentWare Developer Library Referenc*
General Magic Inc., Sunnyvale, Kalifornien USA 15. Juli 1996

[DevGuide96] *Tabriz AgentWare Developer's Guide*
General Magic Inc., Sunnyvale, Kalifornien USA 31. Juli 1996

[DevLib96] *Telescript Developer Libraries Reference*
General Magic Inc., Sunnyvale, Kalifornien USA 31. Juli 1996

[DieZeit97] *Im Jahr der Banane*
In: DIE ZEIT Nr. 3, Hamburg, 10. Januar 1997

[GenMagic] *General Magic WWW Server*
WWW: `http://www.genmagic.com/`

[mobilis] *mobilis: the mobile computing lifestyle magazine*
WWW: `http://www.volksware.com/mobilis/`

[OpGuide96] *Tabriz AgentWare Operator's Guide*
General Magic Inc., Sunnyvale, Kalifornien USA 30. Juni 1996

[PassActive96] *Tabriz: Transforming Passive Networks into an Active, Persistent, and Secure Business Advantage*
General Magic White Paper
General Magic Inc., Sunnyvale, Kalifornien USA 30. Juli 1996

[ProgGuide96] *Telescript Programming Guide*
General Magic Inc., Sunnyvale, Kalifornien USA 15. Juli 1996

[SaveSec96] *An Introduction to Sefety and Security in Telescript*
General Magic Inc., Sunnyvale, Kalifornien USA 1996

[Sony] *Ein WWW Server von Sony Electronics Inc.*
WWW: `http://www.sel.sony.com/SEL/Magic/`

[White94a] White, James E.: *Telescript Technology: The Foundation for the Elektronic Marketplace*
General Magic White Paper
General Magic Inc., Sunnyvale, Kalifornien USA 1994

[White94b] White, James E.: *Telescript Technology: Scenes from the Elektronic Marketplace*
General Magic White Paper
General Magic Inc., Sunnyvale, Kalifornien USA 1994

[White95a] White, James E.: *Mobile Agents*
In *Software Agents*, Brandshaw, Jeffrey (ed.), AAAI Press and MIT Press, Menlo Park, Kalifornien USA 1996

[White95b] White, James E.: *Telescript Language Reference Manual*
General Magic Inc., Sunnyvale, Kalifornien USA Oktober 1995

[White95c] White, James E.: *Telescript Technology: An Introduction to the Language*
General Magic White Paper
General Magic Inc., Sunnyvale, Kalifornien USA Oktober 1995

Die von General Magic veröffentlichte Literatur ist (seit Januar 1997 eingeschränkt) auch auf dem Webserver von General Magic zu finden [GenMagic] und auf ihrem Ftpserver `ftp.genmagic.com`.