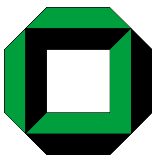
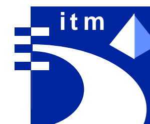
View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by KITopen



Universität Karlsruhe
Fakultät für Informatik
Institut für Telematik
76128 Karlsruhe



Architektur vernetzter Systeme

Seminar Wintersemester 1999/2000

Herausgeber:
Rainer Ruggaber
Dr. Jochen Seitz

Universität Karlsruhe
Institut für Telematik

Interner Bericht 2000-10
ISSN 1432 - 7864

Inhaltsverzeichnis

Vorwort iii

Marcus Denker:
Event und Notification Service in CORBA 1

Fenghui Chen:
Asynchroner Nachrichtenaustausch 15

Olaf Titz:
Sicherheitsmechanismen in CORBA 31

Dennis Koslowski:
Transportprotokolle für drahtlose und mobile Endgeräte 45

Vorwort

Neben der Weiterentwicklung der Kommunikationsinfrastruktur erfolgt parallel eine durch neue Einsatzgebiete getriebene Fortentwicklung im Middleware- und Anwendungsbereich. Der hier vorliegende Seminarband entstammt der Reihe „Architektur vernetzter Systeme" und betrachtet Teilaspekte von CORBA, die für den Einsatz in drahtlosen und mobilen Umgebungen hilfreich und notwendig sind. Hierzu zählen zum einen Aufrufsemantiken, die die enge Bindung von Client und Server durch synchrone Aufrufe aufheben. Zum anderen sind dies die notwendigen Sicherheitsmechanismen, die insbesondere bei der Unterbeauftragung von Servern interessant sind. Abschließend wird ein Protokoll vorgestellt, das auf die speziellen Bedürfnisse drahtloser und mobiler Kommunikation hin entwickelt wurde und auch im CORBA-Umfeld zu einer besseren Leistungsfähigkeit führen kann.

Event und Notification Service in CORBA
Der CORBA Event Service hebt die enge Kopplung von Client und Server auf. Die Kommunikation findet auf Basis von Ereignissen (Events) statt, die von einem Erzeuger (Supplier) zu einem Verbraucher (Consumer) gesendet werden. Erzeuger und Verbraucher sind über einen Ereigniskanal (Event Channel) gekoppelt. Um einige Nachteile des Event Service zu umgehen, hat die OMG als Erweiterung den Notification Service definiert.

Asynchroner Nachrichtenaustausch
Ein asynchroner Nachrichtenaustausch in CORBA ist insbesondere für Anwendungen im drahtlosen Umfeld notwendig. Hiermit wird eine Unabhängigkeit entfernter Aufrufe von bestehenden Transportverbindungen erreicht. Der Time-Independent-Invocation-Modus erlaubt durch die Zwischenspeicherung von Aufrufen, die enge Kopplung von Client und Server bei synchronen Aufrufen aufzuheben und die knappen Ressourcen auf der drahtlosen Strecke effizient auszunutzen.

Sicherheitsmechanismen
Der CORBA Security Service enthält sowohl anwendungstransparente wie auch anwendungsspezifische Sicherheitsmechanismen. Die verschiedenen Mechanismen sind in Ebenen gruppiert, die eine steigende Komplexität aber auch Sicherheit zur Folge haben. Von besonderem Interesse sind dabei die Authentifikation unter CORBA und die Delegation von Sicherheitsattributen bei der mehrstufigen Unterbeauftragung.

Transportprotokolle für drahtlose und mobile Endgeräte
TCP/IP als Standard-Transportprotokoll für CORBA ist für den Einsatz im drahtlosen Umfeld wenig geeignet. Speziell die Staukontrolle durch den Slow-Start-Mechanismus führt auf schmalen Leitungen mit hohen Fehlerraten zu ungenügendem Durchsatz. Neben einigen Erweiterungen von TCP/IP, die diese Nachteile zu beheben versuchen,

existiert mit dem Wireless Application Protocol (WAP) ein speziell für diesen Einsatzbereich konzipiertes Protokoll.

Bevor nun die einzelnen Ausarbeitungen der Seminarbeiträge präsentiert werden, möchten wir allen beteiligten Studenten für ihre engagierte Mitarbeit danken, ohne die weder der Erfolg des Seminars noch die Anfertigung des vorliegenden Berichts möglich gewesen wäre. Hierzu haben auch die nach den einzelnen Vorträgen stattfindenden Diskussionen maßgeblich beigetragen.

Karlsruhe, im April 2000

Rainer Ruggaber Jochen Seitz

Event und Notification Service in CORBA

Marcus Denker

Kurzfassung

Mittels CORBA können verteilte Applikationen einfach über ein Netzwerk miteinander kommunizieren. Doch das von CORBA zur Verfügung gestellte Kommunikationsmodell reicht in manchen Fällen nicht aus, es wird ein Modell zur asynchronen Kommunikation zwischen lose gekoppelten Objekten benötigt. Der *CORBA Event Service* ist ein Versuch, ein solches Kommunikationsmodell bereitzustellen. Der Event Service hat sich aber als unzureichend herausgestellt. Daher wurde er um einige Aspekte zum *Notification Service* erweitert.

1 Einleitung

1.1 Wozu Events?

Ein zentraler Aspekt der objektorientierten Programmierung ist die Kommunikation zwischen den Objekten mittels Nachrichten. Alan Kay, einer der Erfinder des objektorientierten Paradigmas, bezeichnet die Nachrichtenübermittlung sogar als die zentrale Idee von OOP [Kay97].

In den meisten existierenden objektorientierten Systemen sind Nachrichten synchron, das heißt, daß der Sender der Nachricht so lange blockiert, bis der Empfänger die Aufgabe erledigt hat und evtl. ein Ergebnis zurückliefert. Eine Nachricht in einem solchen System ist auch immer an einen einzigen bestimmten Empfänger gerichtet.

Beispiele für solch synchrone Nachrichten sind einmal die Methodenaufrufe der objektorientierten Programmiersprachen, aber auch das von CORBA breitgestellte Request/Response Model (Abbildung 1).

Der Klient (Dienstnehmer) ruft eine Operation des Zielobjektes auf, das sich auch auf einem anderen Rechner befinden kann. Der Dienstnehmer blockiert so lange, bis der Dienstgeber seine Aufgabe erledigt hat und evtl. ein Ergebnis zurückliefert.

In der Praxis hat sich diese einfache synchrone eins-zu-eins Kommunikation als nicht ausreichend herausgestellt. Daher wurde versucht, in CORBA auch andere Kommunikationsmodelle zur Verfügung zu stellen. Der *Event Service* und seine spätere Erweiterung, der *Notification Service*, sind Beispiele für ein solches Kommunikationsmodell.

Ein Event-Service bietet folgende Erweiterungen:

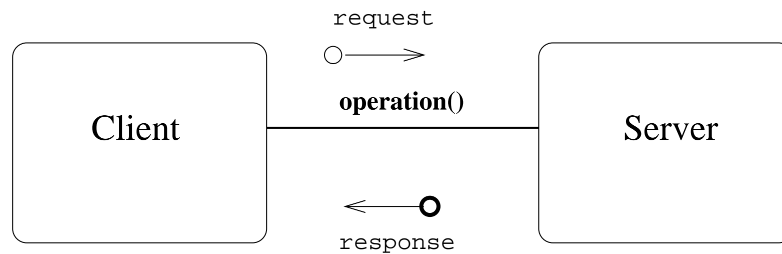


Abbildung 1: Das CORBA Request/Response Modell

- Er entkoppelt die Kommunikation, d.h. die Objekte müssen nichts mehr voneinander wissen, können aber trotzdem Nachrichten austauschen.
- Es können Nachrichten $1:n$ und auch $m:n$ ausgetauscht werden. Eine $1:n$ Kommunikation findet statt, wenn eine Nachricht von einem Sender an viele Empfänger gesendet wird. Die $m:n$ Kommunikation erweitert noch die Anzahl der Sender, d.h. es senden m verschiedene Sender an die gleichen n Empfänger.
- Die Übertragung ist asynchron. Der Sender blockiert nicht.

1.2 Beispiele

Es folgen einige Beispiele um zu verdeutlichen, in welchen Situationen ein solches Event-System eingesetzt werden kann.

- Eine Netzwerkverwaltungsprogramm ist an einer Benachrichtigung interessiert, wenn der Speicherplatz auf einer Festplatte eines Rechners im lokalen Netz erschöpft ist. Dabei können im Betrieb neue Platten und auch weitere Verwaltungsprogramme hinzugefügt werden.
- Eine Anwendung möchte gerne informiert werden, wenn sich die von ihr benötigten Daten ändern. Zum Beispiel möchte ein CASE-Tool benachrichtigt werden, wenn der mit dem CASE-Tool bearbeitete Quellcode von anderen Tools verändert wurde. Das CASE-Tool kann dann den veränderten Code analysieren und seine internen Strukturen auf einen aktuellen Stand bringen.
- Ein Börsenkurs-Programm möchte die Kurse an die angeschlossenen Rechner weitergeben, sobald neue Daten zur Verfügung stehen. Dabei können zur Laufzeit Klienten hinzugeschaltet oder abgemeldet werden.

1.3 Weiteres Vorgehen

Die Seminararbeit gliedert sich in 2 Teile.

Im ersten Teil wird der in CORBA implementierte Event Service vorgestellt. Nach einem Überblick und genauerer Erläuterung der Architektur des Event Service folgt ein Beispiel anhand dessen die genaue Verwendung verdeutlicht wird. Anschließend werden die Probleme analysiert, die zu einigen Erweiterungen des Event Service geführt haben.

Der zweite Teil erläutert nun diese Erweiterungen, wie sie für einen CORBA Notification Service vorgeschlagen werden.

2 CORBA Event Service

2.1 Suppliers/Comsumers

Der CORBA Event Service definiert für Objekte zwei Rollen: „Supplier“ und „Consumer“ (Erzeuger und Verbraucher).

Der Event Supplier erzeugt Ereignisse, der Consumer „verbraucht“ d.h. verarbeitet Ereignisse.

Die Ereignisse werden mittels normaler CORBA-Requests zwischen Erzeuger und Verbraucher ausgetauscht. Die Ereignisse selber sind dabei nicht als CORBA-Objekte implementiert, da das verteilte Objekt Modell von CORBA es nicht erlaubt, Objekte als Parameter zu übergeben (call by value).

2.2 Kommunikationsmodelle

Erzeuger und Verbraucher von Ereignissen können beide sowohl eine aktive als auch eine passive Rolle einnehmen.

2.2.1 Push-Modell

Wenn das *Push-Modell* verwendet wird, initiiert der Erzeuger die Kommunikation. Der Erzeuger sendet das Ereignis zum Verbraucher durch Aufruf seiner `push()`-Operation (siehe Abbildung 2).

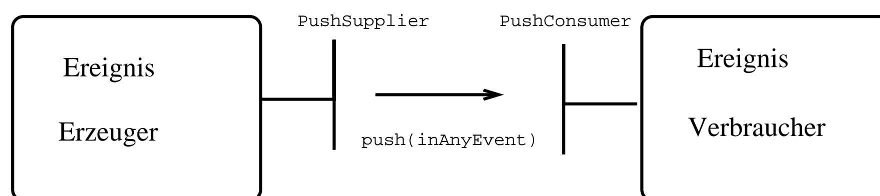


Abbildung 2: Das *Push-Modell*

2.2.2 Pull-Modell

Im Gegensatz dazu ist beim *Pull-Modell* der Verbraucher aktiv: Er ruft die `pull()`-Operation des Erzeugers auf, die ein Ereignis zurückliefert, falls es vorhanden ist (siehe Abbildung 3). Der Erzeuger bietet sowohl blockierende als auch nichtblockierende Varianten der `pull()`-Operation.

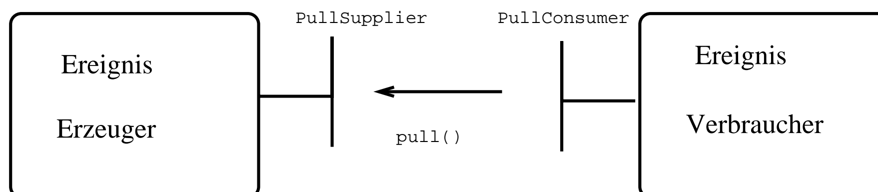


Abbildung 3: Das *Pull-Modell*

2.3 Event Channel

Der Event Channel spielt eine zentraler Rolle innerhalb des Event Service. Er entkoppelt Erzeuger und Verbraucher und ermöglicht Gruppenkommunikation.

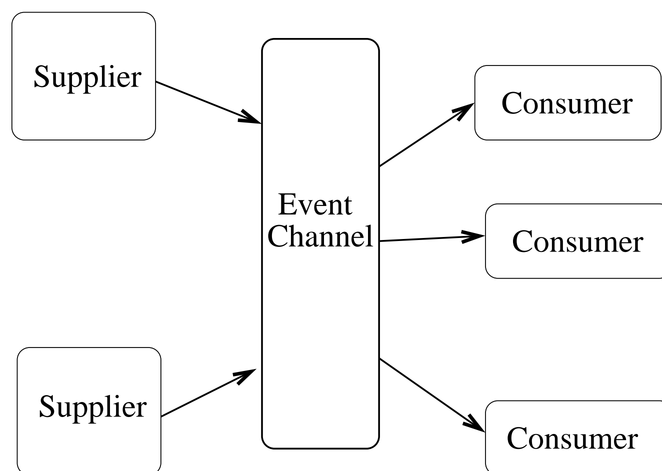


Abbildung 4: Der Event Channel

Ein Event Channel ist im Prinzip ein Objekt, das Ereignisse von mehrereren Erzeugern empfängt und diese an mehrere Emfänger weiterleitet.

Der Event Channel erlaubt einem Erzeuger, Ereignisse an alle interessierten Verbraucher weiterleiten, ohne daß er genau wissen muß, an welche Verbraucher und an wieviele insgesamt das Ereignis weitergeleitet wird. Die Anzahl der Verbraucher kann sich sogar dynamisch zur Laufzeit ändern.

Eine weitere Eigenschaft des Event Channels ist die Umsetzung vom Push-Modell ins Pull-Modell. Das heißt, daß z.B. ein Verbraucher die Ereignisse per Pull-Modell beim Event Channel abholen kann, obwohl der Erzeuger sie mittels Push-Modell angeliefert hat.

2.4 Die IDL-Schnittstellen des Event Services

Wie alle Schnittstellen der OMG *Common Object Services* (COS) ist auch der Event Service mittels OMG IDL definiert.

2.4.1 Das CosEventComm-Modul

Das CosEventComm Modul definiert die Schnittstellen der Erzeuger und Verbraucher. Es existieren sowohl für Erzeuger als auch Verbraucher jeweils zwei Schnittstellen, eine für das Push- und Pull-Modell.

```
module CosEventComm
{
  exception Disconnected {};

  interface PushConsumer {...};

  interface PushSupplier {...};

  interface PullSupplier  {...};

  interface PullConsumer {...};
};
```

Wenn das Push-Modell verwendet werden soll, muß der Ereigniserzeuger die PushSupplier-Schnittstelle implementieren, der Verbraucher implementiert die PushConsumer-Schnittstelle.

```
interface PushConsumer {
   void push (in any data) raises(Disconnected);
   void disconnect_push_consumer();
};
```

Die PushConsumer-Schnittstelle definiert zwei Methoden, die vom Erzeuger aufgerufen werden können. Zum einen die push()-Operation, mit der Ereignisse übergeben werden, zum anderen kann der Verbraucher mit der disconnect_push_consumer()-Methode dazu gebracht werden, die Verbindung aufzugeben.

Sollte die Verbindung beim Aufruf von push() schon unterbrochen sein, wird eine Disconnected-Exception ausgelöst.

```
interface PushSupplier {
     void disconnect_push_supplier();
};
```

Der Erzeuger muß bei Verwendung des Push-Modells einzig die Methode disconnect_push_supplier() implementieren.

Im Pull-Modell wird die aktive Rolle vom Verbraucher eingenommen, also ist entsprechen zum PushSupplier einzig eine disconnect_pull_consumer()-Operation zu definieren.

```
interface PullConsumer {
   void disconnect_pull_consumer();
};
```

Der Verbraucher kann nun die vom Erzeuger zur Verfügung gestellten Methoden aufrufen, im einzelnen `pull()`, `try_pull()` und `disconnect_pull_supplier()`.

```
interface PullSupplier {
   any pull() raises(Disconnected);
   any try_pull(out boolean has_event)
     raises(Disconnected);
   void disconnect_pull_supplier();
};
```

Der Verbraucher kann Ereignisse sowohl durch Aufruf von `pull()` oder `try_pull()` abrufen. Dabei blockiert die `pull()`-Operation, bis ein Ereignis vorhanden ist. Die `try_pull()`-Operation blockiert nicht. Wenn ein Ereignis verfügbar ist, wird es zurückgeben und der Parameter `has_event` wird auf `true` gesetzt. Sollte kein Event vorhanden sein, so hat `has_event` den Wert `false` (Der Rückgabewert ist in diesem Fall undefiniert).

Sollte die Kommunikation schon unterbrochen worden sein, wird in beiden Fällen die `Disconnected`-Exception ausgelöst.

2.4.2 Das `CosEventChannelAdmin`-Modul

Dieses Modul definert die Schnittstellen zum Event Channel.

Zum einen enthält dieses Modul alle Schnittstellen, die Erzeuger und Verbraucher benötigen, um eine Verbindung zum Event Channel aufzubauen. Zum anderen sind hier die Schnittstellen definiert, über die Ereignisse mit dem Event Channel ausgetauscht werden.

```
interface EventChannel {
    ConsumerAdmin for_consumers();
    SupplierAdmin for_supplier();
    void destroy();
};
```

Mittels der `destroy()`-Methode kann der Event Channel zerstört werden. Daneben wird jeweils für Erzeuger und Verbraucher eine Methode angeboten: `for_consumers()` und `for_suppliers()`. Diese geben ein `ComsumerAdmin`- bzw. `SupplierAdmin`-Objekt zurück:

```
interface ConsumerAdmin {
    ProxyPushSupplier obtain_push_supplier();
    ProxyPullSupplier obtain_pull_supplier();
};
```

```
interface SupplierAdmin {
    ProxyPushConsumer obtain_push_consumer();
    ProxyPullConsumer obtain_pull_consumer();
};
```

Der `ConsumerAdmin` bzw. `SupplierAdmin` bietet nun Methoden an, mit denen das entsprechende Objekt die Referenzen ihres Gegenparts innerhalb des Event Channel bekommen kann. Wenn ein Erzeuger bei dem Event Channel Ereignisse abliefern möchte, so muß der Event Channel ein Interface bereitstellen, das dem eines Verbrauchers entspricht. Entsprechend muß für einen Verbraucher eine Erzeuger-Schnittstelle vorhanden sein.

Für diesen Zweck werden vier sogenannte *Proxy*-Schnittstellen definiert. Diese erben von den entsprechenden Schnittstellen im `CosEventComm`-Modul, fügen diesen aber jeweils noch eine Methode hinzu. Mit dieser Methode, z.B. `connect_push_supplier()` bei der Schnittstelle `ProxyPushConsumer`, stellt das Objekt die Verbindung mit dem Event Channel her.

```
interface ProxyPushConsumer: CosEventComm::PushConsumer {
   void connect_push_supplier(
           in CosEventComm::PushSupplier push_supplier)
      raises(AlreadyConnected);
  };

interface ProxyPullSupplier: CosEventComm::PullSupplier {
  void connect_pull_consumer(
         in CosEventComm::PullConsumer pull_consumer)
      raises(AllreadyConnected);
};

interface ProxyPullConsumer: CosEventComm::PullConsumer {
  void connect_pull_supplier(
         in CosEventComm::PullSupplier pull_supplier)
      raises(AlreadyConnected, TypeError);
};

interface ProxyPullSupplier: CosEventComm::PushSupplier {
  void connect_push_consumer(
         in CosEventComm::PushConsumer push_consumer)
      raises(AlreadyConnected, TypeError);
};
```

2.5 Beispiel

Um das Zusammenspiel von Erzeuger, Verbraucher und Event Channel zu verdeutlichen soll ein einfaches Beispiel vorgestellt werden, bei dem ein Erzeuger über einen Event Channel mit einem Verbraucher kommuniziert. Sowohl Erzeuger als auch Verbraucher verwenden das Push-Modell.

2.5.1 Verbindungsaufbau

Bevor Ereignisse ausgetauscht werden können, müssen sich Erzeuger und Verbraucher mit dem Event Channel verbinden. Der Verbraucher muß dazu folgende drei Schritte durchführen:

1. Der Verbraucher ruft die `for_consumers()`-Methode des Eventchannel auf, um eine Referenz auf ein `CunsumerAdmin`-Objekt zu bekommen.
2. Nachdem die `for_consumers()`-Methode aufgrufen wurde, kann nun eine Methode des `ConsumerAdmin`-Objektes aufgerufen werden. Entsprechend dem gewählten Kommunikationsmodell (*push* oder *pull*) ruft der Verbaucher entweder `obtain_pull_supplier()` oder `obtain_push_supplier()` auf, um eine entsprechende Referenz auf ein Proxy-Objekt zu bekommen.
3. Wenn der entprechende Erzeuger-Proxy bekannt ist, kann sich der Verbraucher mit dessen Hilfe mit dem Event Channel verbinden. Dazu existiert die Methode `connect_push_consumer()`. Als Parameter übergibt er eine Referenz auf sich selbst (`this`).

Abbildung 5 stellt noch einmal alle Schritte dar, die für den Verbindungsaufbau benötigt werden.

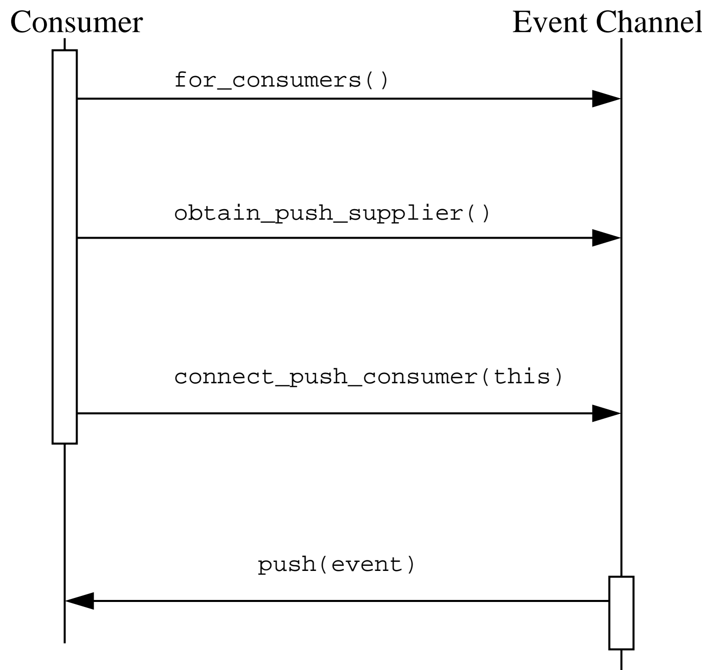


Abbildung 5: Der Verbindungsaufbau

Der Verbindungsaufbau zwischen Erzeuger und Event Channel ist symmetrisch zu dem oben Beschriebenen: Er ruft zuerst die `for_suppliers()`-Methode des Event Channel auf und erhält eine Referenz auf einen `SupplierAdmin`. Dessen `obtain_push_cosumer()`-Methode liefert einen Verbraucher-Proxy, mit dessen Hilfe (Methode `connect_push_supplier()`) sich der Erzeuger mit dem Event Channel verbindet.

2.5.2 Event-Kommunikation

Nachdem sowohl Erzeuger als auch Verbraucher die Verbindung zum Event Channel aufgebaut haben, können sie damit beginnen, miteinander zu kommunizieren.

Die zwischen Erzeuger und Verbraucher ausgetauschten Ereignisse werden mittels IDL spezifiziert, damit man sie in den CORBA-Typ `Any` konvertieren kann.

Für das Beispiel wurde angenommen, daß einzig das Push-Modell Verwendung findet. Daher wird der Erzeuger das Ereignis mittels der `push()`-Methode an den Event Channel übergeben. Der Event Channel selber wiederum ruft die `push()`-Methode des Verbrauchers auf.

2.6 Typed Events

Neben dem beschriebenen einfachen Event Service bietet CORBA auch eine typisierte Variante („Typed Events“).

Der bis jetzt betrachtete „einfache“ Event Service verlangt, daß die Ereignisse vom CORBA-Typ `Any` sind. Dadurch gehen alle Typinformationen verloren, eine Fehlerquelle entsteht.

Die im CORBA Event Service spezifizierten Typed Events sollten es erlauben, ganze in IDL spezifizierte Schnittstellen über den Ereignis-Mechanismus zu verwenden. Das heißt, der Erzeuger kann z.B. die in dieser Schnittstelle definierten Methoden des Verbrauchers mit Hilfe des Event Channels aufrufen.

In der Praxis hat es sich aber herausgestellt, daß die Spezifikation nicht einfach umsetzbar war: Sie ist (laut [ScVi97a]) nicht in allen ORB implementiert. Eine genaue Beschreibung der Typed Events findet sich in [OMGr97].

2.7 Probleme

Beim Einsatz des CORBA-Event-Systems hat sich herausgestellt, daß einige schwerwiegende Probleme in der realisierten Architekur existieren.

Kein Filtern von Events Ein Verbraucher, der sich bei einem Event Channel anmeldet, erhält alle Events, auch wenn er eigentlich nur ein Teil dieser Events wirklich benötigt.

Netzwerküberlastung Die fehlenden Möglichkeiten zur Eventfilterung führen dazu, daß alle Events an alle Verbraucher geliefert werden müssen. Eine unnötig hohe Netzwerkbelastung ist die Folge.

Quality of Service Es gibt keine Möglichkeit, die Dienstgüte festzulegen. Die Auslieferung von Ereignissen an die Verbraucher sollte sich je nach gewünschter Qualität konfigurieren lassen. Bei verlangter hohe Diestgüte müßte z.B. die Auslieferung an jeden einzelnen Verbraucher garantiert sein, d.h. der Event Channel müßte den Event puffern und die Auslieferung immer wieder versuchen. Falls aber eine geringe Dienstgüte ausreicht, könnte der Event Channel so konfiguriert werden, daß er nur einen Versuch unternimmt, das Ereignis auszuliefern.

3 CORBA Notifikation Service

Die in Kapitel 2.7 erläuterten Probleme des Event Service gaben Anlaß, einen neuen, erweiterten Ereignisdienst zu entwickeln.

Der *CORBA Notification Service* ist eine Erweiterung des CORBA Event Service. Er ist abwärtskompatibel zum Event Service, und auch die Erweiterungen fügen sich in die Konzepte des Event Service ein.

Auf eine ausführliche Beschreibung der IDL-Schnittstelle wird verzichtet. Eine genaue Beschreibung findet sich in [Gree98].

3.1 Überblick

Die wichtigsten Erweiterungen betreffen zwei Bereiche: *Event Filtering* und *Quality of Service*.

Die Verbraucher können den Proxy-Interfaces Filter zuordnen. Filter sind Objekte, die z.B. den einzelnen Verbraucher-Proxies zugeordnet werden und die alle Arten von Ereignisse beschreiben, die zum Verbraucher weitergeleitet werden sollen. Alle anderen werden herausgefiltert.

Unter *Quality of Service* versteht man die Definition von einzuhaltenden Dienstgüten. Der Notifikation Service definiert einige solche Parameter (siehe Kapitel 3.2.1).

Der Event Channel des Notifikation Service erlaubt es, mehrere Admin-Objekte zu definieren. Sowohl Event Filter als auch Dienstgüteparameter eines Admin-Objekts gelten für alle verwalteten Erzeuger und Verbraucher. Daher lassen sich also Objekte mit gleichen Parametern zu Gruppen mit gleichen Filtern und gleicher Dienstgüte zusammenfassen.

Möglich wurden diese Erweiterungen durch die Einführung von *Structured Events*. Diese erlauben es, neben den eigentlichen Ereignisdaten auch weitere Daten zu enthalten, etwa die Dienstgüteparameter.

3.2 Erweiterungen

Im folgenden werden einige der im Notifikation Service vorgenommenen Erweiterungen genauer beschrieben: *Event Filtering* und *Quality of Service*.

3.2.1 Event Filtering

Die Filter-Objekte erlauben es, die Auslieferung von Ereignissen auf genau die Ereignisse einzuschränken, an denen der Empfänger interessiert ist.

Die Filter können durch sog. *Constraints* (Einschränkungen) bestimmen, welche Ereignisse ausgeliefert werden. Constraints werden dadurch definiert, daß jedem Ereignistyp ein Ausdruck zugeordnet wird. Dieser Ausdruck wird in einer definierten Sprache, der *Filtering Constraint Language* formuliert.

Eine genaue Beschreibung dieser Sprache findet sich in [Gree98]. Hier sollen einige kleine Beispiele genügen.

Der Notification Service erlaubt es, Ereignisse durch gewisse Eigenschaften zu beschreiben. So können den Ereignissen z.B. ein Typ zugeordnet und Namen für Ereignisse definiert werden.

- Nehme alle Ereignisse vom Typ `CommunicationsAlarm` an, aber keine `lost_package`-Ereignisse:

  ```
  $Type_name == 'CommunicationsAlarm' and not
  ($event_name == 'lost_packet')
  ```

- Nehme `CommunicationsAlarm` Ereignisse mit einer Priorität größer 5 an:

  ```
  $type_name == 'CommunicationsAlarm' and
  $priority > 5
  ```

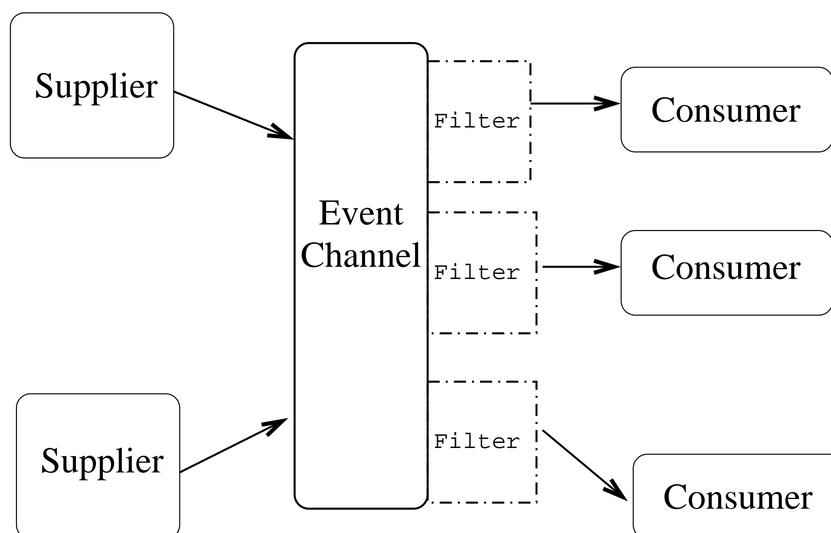


Abbildung 6: Event Channel mit Filtern

3.2.2 Dienstgüte

Der Notification Service erlaubt es, die Dienstgüte der Ereignisauslieferung mittels verschiedener Parameter zu steuern. Dabei kann man diese Parameter nicht nur global für den Notification Channel setzen, sondern auch beschränkt auf einzelne Gruppen von Ereignissen und sogar für einzelne Ereignisse definieren.

Die Dienstgüteparameter können im einzelnen für folgende Objekte gesetzt werden:

- Den komplette Notification Channel.
- Admin-Objekt der Erzeuger und Verbraucher.

- Die Proxy-Objekte.
- Den einzelnen Event.

Setzt man Dienstgüteparameter für ein Admin-Objekt, so gelten sie für alle über dieses AdminObjekt erzeugte Proxy-Objekte. Eine für ein Proxy-Objekt definierte Dienstgüte gilt dann für alle über diesen Proxy laufenden Ereignisse.

Es folgt eine kurze Beschreibung der wichtigsten Dienstgüteparameter.

Reliabiliy Die Verläßlichkeit kann auf zwei Ebenen geregelt werden: Einmal für den Kanal und einmal für das einzelne Ereignis. Siehe [Gree98], Seite 53.

Priority Normalerweise definiert der Notification Service keine Reihenfolge für die Auslieferung. Bei Angabe einer Priorität (zwischen -32767 und $+32767$) versucht der Event Channel die Ereignisse in Reihenfolge dieser Prioritäten auszuliefern.

Expiry Time Es können verschiedene Werte für die zeitliche Gültigkeit konfiguriert werden: `StopTime` definiert einen exakten Zeitpunkt, ab dem das Ereignis ungültig ist. `TimeOut` dagegen definiert diesen Zeitpunkt relativ zur aktuellen Zeit.

Earliest Delivery Time Durch den Wert `StartTime` kann festglegt werden, ab wann ein Ereignis ausgeliefert werden darf.

Order Policy Mit diesem Parameter kann festgelegt werden, in welcher Reihenfolge die Ereignisse ausgeliefert werden. Mögliche Werte dieses Parameters sind:

`AnyOrder` Jede Ordnung ist erlaubt.

`FifoOrder` Auslieferung in Reihenfolge der Einlieferung.

`PriorityOrder` Auslieferung bestimmt durch die Prioritäten.

`DeadlineOrder` Die *Expiry Time* legt die Auslieferung fest: Als erstes werden die Ereignisse ausgeliefert, die als erstes ungültig würden.

Discard Policy Daduch kann festgelegt werden, in welcher Reihenfolge bei einer Überlastung Ereignisse gelöscht werden können.

4 Zusammenfassung

Der CORBA Notification Service erweitert die Möglichkeiten der Kommunikation zwischen Objekten. Das normale Request/Response Modell wird um eine Möglichkeit zur asynchronen, lose gekoppelten $m : n$ Kommunikation ergänzt, die in der Praxis häufig benötigt wird. Die Probleme, die beim Event Service anfänglich vorhanden waren, wurden durch die Erweiterungen des Notification Service beseitigt.

Literatur

[Gree98] M.J. Greenberg (Hrsg.). *Notification Service – Joint Revised Submission.* OMG. November 1998.

[Kay97] Alan Kay. *The Computer Revolution Hasn't Happend Yet! Keynote OOPSLA 1997.* UVC Video. Oktober 1997.

[OMGr97] Object Management Group (Hrsg.). *Event Service Specification*, Kapitel 4. OMG. 1997.

[ScVi97a] D.C. Schmidt und S. Vinoski. Object Interconnections – Overcoming Drawbacks in the OMG Events Service. *SIGS C++ Report Magazine*, Februar 1997.

[ScVi97b] D.C. Schmidt und S. Vinoski. Object Interconnections – the OMG Events Service. *SIGS C++ Report Magazine*, Februar 1997.

[Tech98] IONA Technologies. *Notification White Paper.* Mai 1998.

Abbildungsverzeichnis

1 Das CORBA Request/Response Modell 2

2 Das *Push-Modell* 3

3 Das *Pull-Modell* 4

4 Der Event Channel 4

5 Der Verbindungsaufbau 8

6 Event Channel mit Filtern 11

Asynchroner Nachrichtenaustausch

Fenghui Chen

Kurzfassung

CORBA ist ein von dem Object Management Group (OMG) entwickelter Standard zur Kommunikation in verteilten Systemen. Als Kommunikationsmiddleware spielt CORBA eine wichtige Rolle in verteilten Umgebungen. Erste CORBA Version unterstützten als Kommunikationsparadigma den synchronen Nachrichtenaustausch und einen schwachen Asynchronen Nachrichtenaustausch. Als Erweiterung wird der Mechanismus des asynchronen Nachrichtenaustauschs (Asynchronous Method Invocation - AMI) in der CORBA 3.0 Spezifikation eingeführt. Dieser Beitrag dient dazu, die Architektur, die Funktionsweise und die Vorteile des asynchronen Nachrichtenaustauschs in der CORBA 3.0 Spezifikation zu beschreiben und dessen Probleme zu diskutieren. Am Ende wird ein Anwendungsprototyp von AMI vorgestellt.

Schlüsselworte: CORBA, Client-Objekt, Ziel-Objekt, AMI, Polling Modell, Callback Modell, TII, Client-Router, Ziel-Router, Store and Forward.

1 Motivation

Die Common Object Request Broker Architecture (CORBA) ist momentan die wichtigste Middleware für verteilte Anwendungen. CORBA wird als ein Industrie Standard von der OMG (Object Management Group) entwickelt. Erste CORBA Versionen hatten noch viele Schwächen bei der Spezifikation des Nachrichtenaustauschs. Zunächst werden diese Schwächen aufgeführt.

1.1 Modelle des Nachrichtenaustauschs in CORBA 2.0

In den Versionen vor CORBA 3.0 sind nur drei Modelle des Nachrichtenaustauschs vorgesehen. Diese sind der synchrone bidirektionale Aufruf, der unidirektionale Aufruf und der synchrone abgekoppelte bidirektionale Aufruf.

Der synchrone bidirektionale Aufruf: Der Client sendet eine Anfrage an das Ziel-Objekt und warte auf die Antwort. Während der Client wartet, wird der Client-Thread gesperrt. Das Modell wird in Abbildung 1 gezeigt.

Der unidirektionale Aufruf: Der Client sendet nur eine Anfrage und erwartet keine Antwort. Die Client-Anwendung wird nicht blockiert. CORBA übernimmt keine Garantie für die Auslieferung der Anfrage.

Der synchrone abgekoppelte bidirektionale Aufruf: In diesem Modell hat der Client zwei Möglichkeiten. Zum einen kann der Client eine Anfrage an das Ziel-Objekt senden und dann die lokale Bearbeitung fortsetzen. Der Client-Thread wird nicht gesperrt. Der Client kann zu einem späteren Zeitpunkt überprüfen, ob das Ziel-Objekt die Antwort bereits zurückgeschickt hat. Zum anderen kann der Client eine Anfrage an das Ziel-Objekt schicken und dann auf die Antwort warten, der Client und somit auch die lokale Bearbeitung werden gesperrt wie beim synchronen bidirektionalen Aufruf.

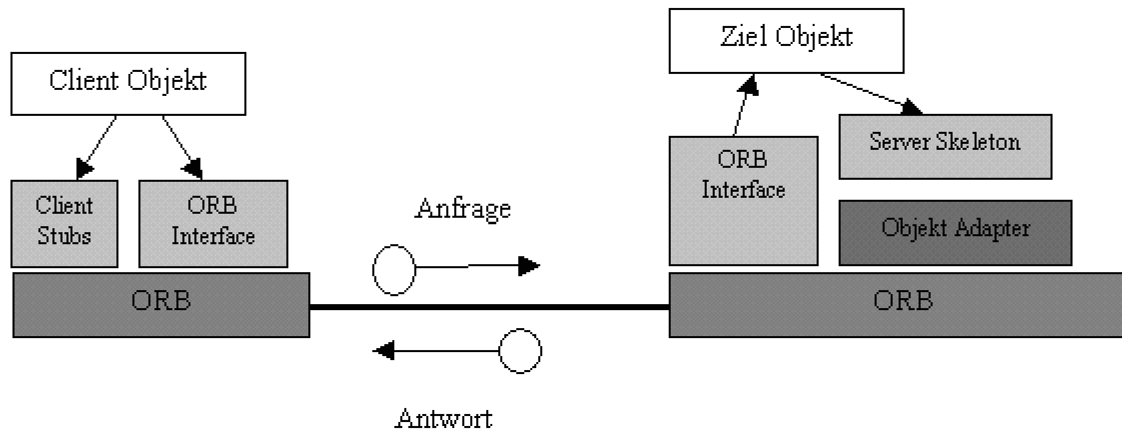


Abbildung 1: Der synchroner bidirektionale Nachrichtenaustausch

1.2 Probleme

Die Modelle des Nachrichtenaustauschs vor CORBA 3.0 haben folgende Schwächen:

- Geringe Leistungsfähigkeit: Bei dem synchronen bidirektionalen Nachrichtenaustausch wird der Client-Thread gesperrt, während der Client auf die Antwort des Ziel-Objekts wartet. Falls diese Anwendung (Client-Programm) auf einem mobilen Gerät z. B. Laptop läuft und das Laptop über eine drahtlose Verbindung an das Festnetz angebunden ist, muß diese Verbindung während der Anfragebearbeitung erhalten bleiben.

- Einschränkungen beim synchronen abgekoppelten bidirektionale Aufruf: Dieses Modell kann nur über das Dynamic Invocation Interface (DII) aufgerufen werden. Die Erfahrung hat gezeigt, daß DII-basierte Anwendungen mehr Code als SII-basierte Anwendungen benötigen und relativ schwieriger zu programmieren sind. Im Gegensatz dazu sind Static Invocation Interface (SII)-basierte Anwendungen leicht zu schreiben und zu warten. Ein weiterer Vorteil ist, daß SII sicherer bei Datentypumwandlungen ist.

- Unidirektionale Aufrufe können nicht garantieren, daß der Client-Thread nicht gesperrt wird während der Client eine Anfrage sendet. Das Senden einer Anfrage in diesem Modus verwendet ein Übertragungsprotokoll wie z. B. IIOP (Internet Inter-ORB Protokoll). IIOP hängt von TCP/IP ab. Die Transportschicht kann bewirken, daß der Client-Thread gesperrt wird. Andererseits können unidirektionale Aufrufe nicht garantieren, daß die Anfrage das Ziel-Objekt erreicht hat.

Für obige Probleme bietet CORBA keine Lösung an. Die aktuelle CORBA Spezifikation Version 3.0 führt den asynchronen Nachrichtenaustausch (Asynchronous Methode Invocation - AMI) ein. Dieser soll einige der dargestellten Probleme lösen.

2 Architektur des asynchronen Nachrichtenaustauschs in CORBA

In diesem Abschnitt wird die Architektur des Asynchronen Nachrichtenaustauschs [D. C98a] in CORBA beschrieben. Zunächst wird eine Übersicht über die neuen Möglichkeiten des Nachrichtenaustauschs in CORBA gegeben. Der synchrone Nachrichtenaustausch wurde bereits im 1. Abschnitt erläutert.

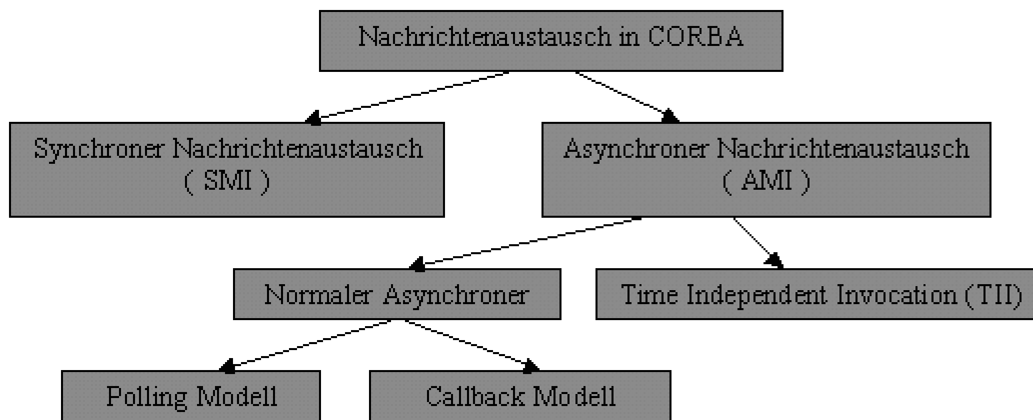


Abbildung 2: Klassifikation des CORBA Nachrichtenaustauschs

Wie Abbildung 2 zeigt existieren zwei Möglichkeiten des asynchronen Nachrichtenaustauschs. Der eine ist der normale asynchrone Modus, der andere ist der Time Independent Invocation (TII) Modus.

2.1 Normaler Asynchroner Nachrichtenaustausch in CORBA

Beim normalen AMI wird das Polling Modell und das Call Back Modell unterschieden.

- Das Polling Modell
  Das Polling Modell hat die folgende Architektur wie Abbildung 3 zeigt.

  In diesem Modell funktioniert AMI wie folgt.
  1) Der Client stellt eine Anfrage, um eine Methode des Ziel-Objekts aufzurufen.
  2) Der Client erhält eine Referenz auf ein Polling-Objekt Poller. Das Polling-Objekt enthält eine Reihe von Funktionen, mit denen der Client den Zustand der Anfragenbearbeitung abfragen oder das Ergebnis der Anfrage abholen kann.
  3) Das Ziel-Objekt schickt das Ergebnis an den Poller.
  4) Der Client ruft eine Methode des Poller auf, um die Ankunft des Ergebnisses abzufragen. Falls das Ziel-Objekt das Ergebnis noch nicht zurückgeschickt hat,

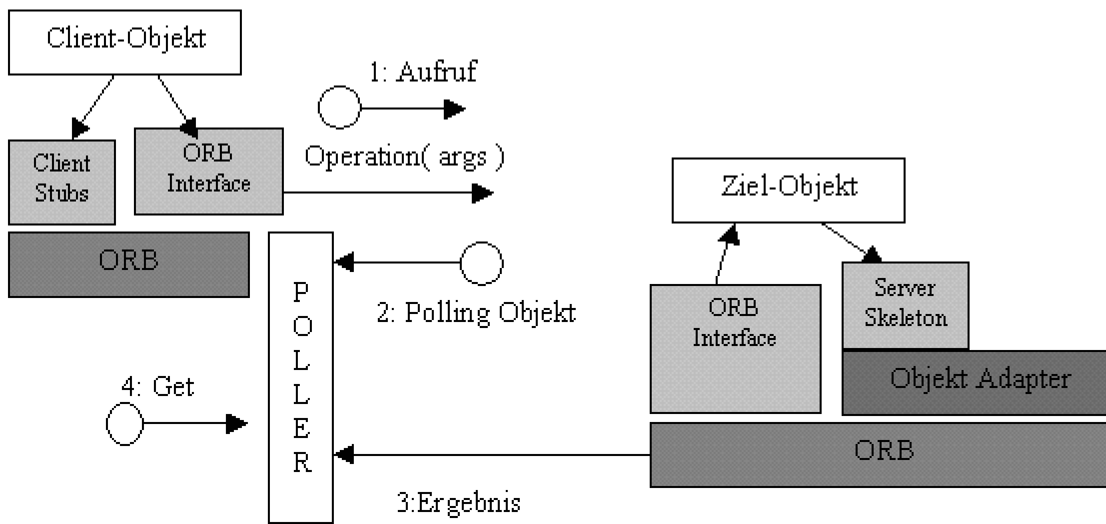


Abbildung 3: Architektur des Polling Modells

kann der Client zum einen auf das Ergebnis warten, so daß der aufrufende Thread gesperrt wird oder zum anderen auf das Warten verzichten, so daß der aufrufende Thread fortgesetzt werden kann. Der Client kann später erneut die Ankunft des Ergebnisses abfragen.
5) Wenn das Ergebnis bereits zurückgeschickt worden ist, kann der Client das Ergebnis durch den Aufruf einer Methode des Poller abholen.

- Callback Modell
  Das Callback Modell hat die folgende Architektur wie Abbildung 4 zeigt.

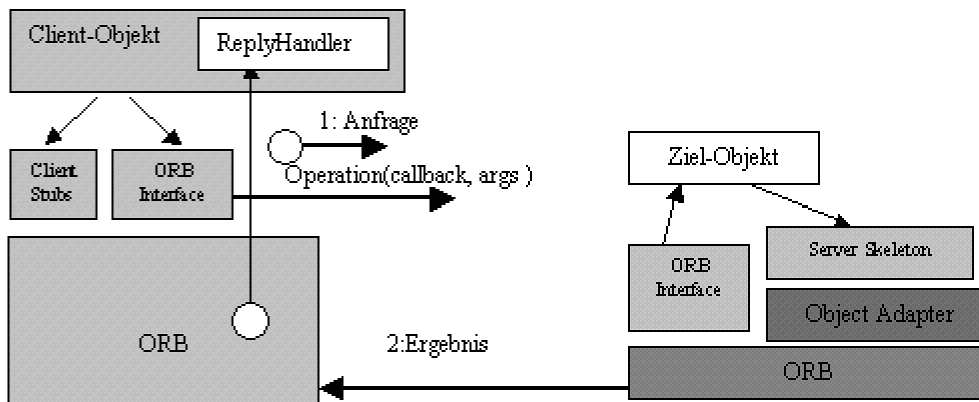


Abbildung 4: Architektur des Callback Modells

Der Methodenaufruf in diesem Modell funktioniert wie folgt.

1) Der Client stellt eine Anfrage, um eine Methode des Ziel-Objekts aufzurufen. Beim Aufruf schickt der Client eine Referenz auf das Objekt ReplyHandler als Parameter mit. Der Replyhandler ist ein Teil des Clients.
2) Der Ziel-Objekt schickt das Ergebnis an den Client-ORB.
3) Wenn der Client-ORB das Ergebnis erhält, ruft der Client-ORB den Reply-Hanlder auf und übergibt dem ReplyHanlder das Ergebnis. Der Client erhält

somit das Ergebnis.

- Vergleich des Polling Modells und des Callback Modells

Modell	Effizienz	Realisierbar
Callback	positiv	negativ
Polling	negativ	positiv

Das Callback Modell ist effizienter als das Polling Modell, da der Client nicht auf das Ergebnis warten muß. Das Polling Modell muß jedoch entweder auf das Ergebnis warten oder regelmäßig die Ankunft des Ergebnisses prüfen. Ein Client im Polling Modell ist ein reiner Client, der nur die Anfrage stellt. Im Callback Modell ist er jedoch auch ein Server, falls der Client-ORB das Ergebnis an den Reply-Hanlder sendet. Der Client-ORB muß die Methode des ReplyHandler aufrufen. Deshalb ist die Realisierung eines solchen Clients relativ schwieriger.

2.2 Time Independent Invocation (TII)

2.2.1 Einführung

TII ist ein spezielles AMI in CORBA, das das Store and Forward von Aufrufen und Ergebnissen ermöglicht. Beim TII Modell kann sich der Client nach dem Absetzen des Aufrufs von dem Netzwerk trennen. Die Anfragen werden in der Warteschlange des Client-Router gespeichert. Der Client-Router leitet die Anfrage an den Ziel-Router weiter. Der Ziel-Router sendet die synchrone Aufrufe an den Server ORB und erhält das Ergebnis. Sobald der Client-Router verfügbar ist, wird das Ergebnis an diesen Client-Router zurückgeschickt und beim Client-Router gespeichert. Der Client kann das Ergebnis beim Client-Router abholen.

2.2.2 Architektur und Ablauf eines TII

Die Architektur des TII wird in Abbildung 5 gezeigt. Beim TII Modell werden zwei neue Komponenten, der Client-Router und der Ziel-Router, als Erweiterung zum normalen AMI eingesetzt. Dem Client-Router und dem Ziel-Router ist gemeinsam, daß sie beide Anfragen empfangen können. Jedoch kann der Client-Router eine Anfrage nur an einen weiteren Client-Router oder einen Ziel-Router schicken. Ein Ziel-Router schickt Anfragen nicht mehr an weitere Router, sondern verteilt die Anfragen auf die verschiedenen Ziel-Objekte. Wie Abbildung 5 zeigt besitzt der Ablauf des TII im wesentlichen sieben Schritte:

(1) Der Client ruft eine Methode des Ziel-Objekts auf. Die Anfrage wird an den Client-ORB weitergeleitet. Zuerst versucht der Client-ORB mit Hilfe des normalen AMI Mechanismus, die Anfrage an das Ziel-Objekt zu senden. Ist zu diesem Zeitpunkt die Verbindung zum Ziel-Objekt nicht möglich, geht der Versuch schief. Der Client-ORB erkennt diesen Fehler und übergibt dem Client jedoch nicht eine Fehlermeldung, sondern entscheidet sich, diese Anfrage an den Client-Router zu senden. Wie der Client-Router durch den Client-ORB gefunden wird, wird im nächsten Abschnitt beschrieben.

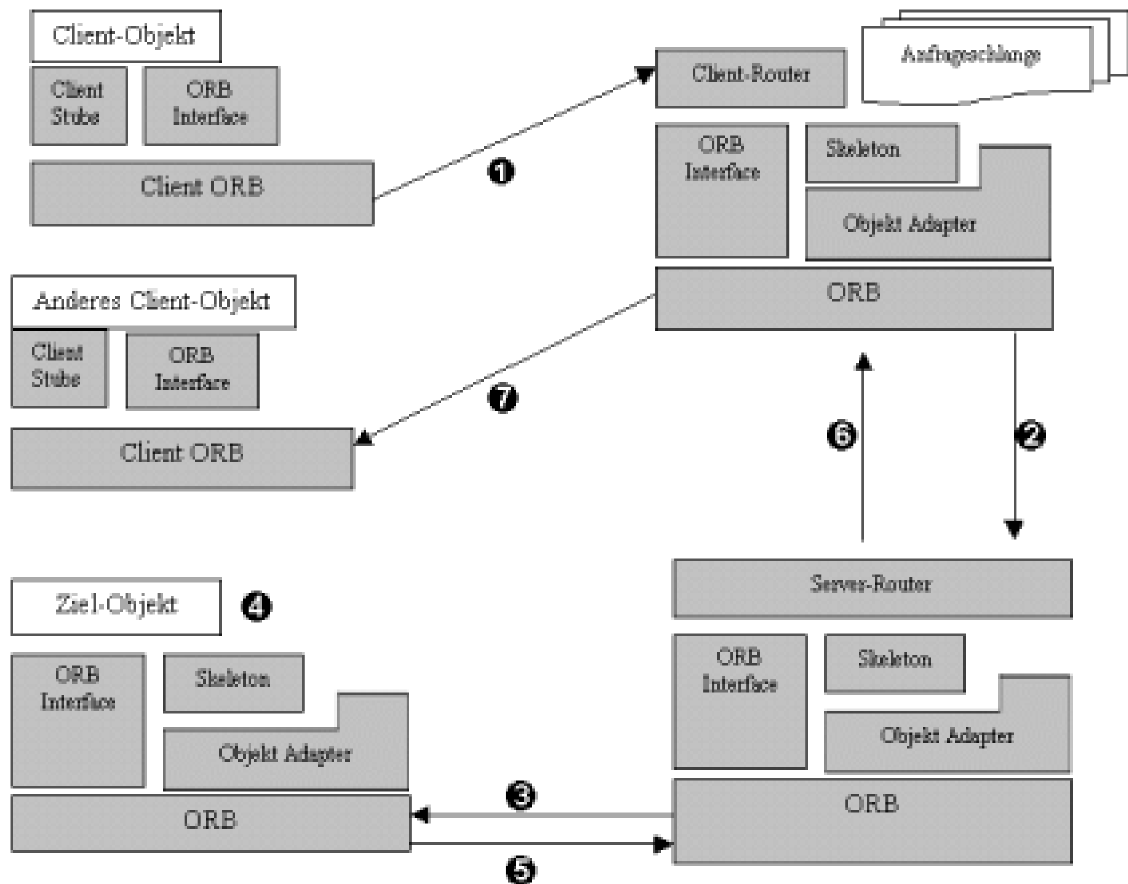


Abbildung 5: Architektur und Ablauf des TII

Nachdem der Client-ORB die Anfrage an den Client-Router gesendet hat, kann sich der Client vom Netzwerk trennen.

(2) Der Client-Router empfängt diese Anfrage und speichert sie in einer Warteschlange. Der Client-Router kann mehrere Anfragen von einem Client oder von mehreren Clients speichern. Sobald ein Ziel-Router erreichbar ist, baut der Client-Router eine Verbindung zu diesem Ziel-Router auf und überträgt die gespeicherten Anfragen.

(3) Der Ziel-Router erhält die Anfrage und macht einen synchronen Aufruf auf dem Server-ORB.

(4) Der Server-ORB erhält einen synchronen Aufruf und ruft die Methode des Ziel-Objekts auf.

(5) Der Server-ORB schickt den Rückgabewert der Methode als Ergebnis an den Ziel-Router.

(6) Der Ziel-Router wandelt das Ergebnis in eine Anfrage um und sendet synchron das Ergebnis an einen Client-Router. Der Client-Router speichert das Ergebnis ab. Details zum Ziel-Router werden in 3.2.1 noch beschrieben. Der Client kann das Ergebnis beim Client-Router abholen.

(7) Die weitere Möglichkeit ist, daß eine andere Anwendung, die den ursprünglichen Aufruf nicht erzeugt hat, das Ergebnis beim Client-Router abholen kann.

Die wichtige Eigenschaft des TII ist das Store and Forward von Nachrichten. Das heißt, die Anfrage des Clients wird zunächst beim Client-Router abgespeichert und dann von

dem Client-Router und einem oder mehreren Ziel-Routern weitergeleitet und bis die Anfrage beim Ziel-Objekt eintrifft. Das Ergebnis der Anfrage wird zunächst beim Ziel-Router gespeichert und dann vom Ziel-Router und Client-Routern weitergeleitet bis der Client das Ergebnis erhält.

2.2.3 Anwendungsgebiet des TII

Eine wesentliche Eigenschaft des TII ist, daß die Anfragen und die Ergebnisse in den Routern zwischengespeichert werden können. Eine Nachricht kann außerhalb des Client-Objekts bzw. des Ziel-Objekts leben. Das ist sinnvoll für die Anwendungen, die auf Geräten laufen, die nur gelegentlich an das Festnetz angeschlossen sind, wie beispielsweise Anwendungen, die im Laptop laufen. Das Laptop ist unterwegs nur per Mobilfunk an das Festnetz angeschlossen. Aus Kostengründen ist ein normales AMI nicht geeignet, da der Client warten muß, unabhängig davon, ob das Polling Modell oder das Callback Modell verwendet wurde. Mittels TII ist es möglich, die Anwendungen zu unterstützen, die trotz einer ungenügenden Netzanbindung Auslieferungsgarantien benötigen. Deshalb muß das Laptop nicht immer an das Netz angeschlossen sein. Die Anwendung (Client Objekt) im Laptop schickt eine Anfrage an den Client-Router (der Client-Router hier ist eine lokale Komponente). Die Anfragen werden in einer Warteschlange des Client-Routers abgelegt. Wenn das Laptop an das Festnetz angeschlossen wird, versucht der Client- Router alle in der Warteschlange befindlichen Anfragen an den Ziel-Router zu schicken, so daß die drahtlose Verbindung sehr intensiv ausgenutzt wird. Alle Ergebnisse der Ziel-Objekte werden lokal abgespeichert. Die Client-Objekte können die Ergebnisse beim Client-Router abholen. Das TII Modell ist sehr sinnvoll für die Anwendungen, deren Zeitanforderung nicht kritisch ist.

2.3 Unterschied zwischen Normalem AMI und TII

Der Unterschied zwischen normalem AMI und TII liegt darin, daß erstens die Länge der Anfrage und zweitens die Lebensdauer der Anfrage und der Ergebnisse im TII und normalem AMI unterschiedlich sind. Wenn der Client-ORB eine Anfrage mit normalem AMI an das Ziel-Objekt, das in diesem Zeitpunkt nicht an Netz angeschossen ist, schickt, versucht der Client-ORB nur für eine bestimmte Zeitdauer (Timeout). Danach erzeugt er einen Fehler und der Client erhält die Fehlermeldung dass das Ziel-Objket nicht erreichtbar ist . Diese Anfrage ist gültig bis diese Fehlermeldung erzeugt wird. Im TII kann eine Anfrage im Client-Router dauerhaft gespeichert werden, bis diese Anfrage den Ziel-Router erreicht. Wenn das Ziel-Objekt nicht erreichbar ist, bleibt die Anfrage jedoch weiter beim Client-Router. Der Client-Router versucht später erneut die Anfrage an den Ziel-Router zu schicken. Der Unterschied ist, daß die Anfrage im TII während der Abwesenheit des Ziel-Objekts überleben kann, im normalen AMI jedoch nicht. Analog kann das Ergebnis auch beim Client-Router dauerhaft gespeichert werden, bis der Client es abholt. Das Ergebnis im TII kann bei der Trennung des Client-Objekts vom Netz überleben, im Gegensatz zum normalen AMI verschwindet das Ergebnis einfach im Netz. Die Lebensdauer der Anfragen und der Ergebnisse im TII ist länger als die Lebensdauer im normalen AMI.

TII ist sinnvoll für Anwendungen, die eine garantierte Auslieferung der Anfrage erfordern. Das normale AMI kann diese Anforderung nicht garantieren und ist somit nicht geeignet für solche Anwendungen.

Modell	Länge der Anfragen	Lebensdauer der Anfragen	Lebensdauer der Ergebnisse	Zusätzliche Komponente
Normales AMI	Standard Anfrage	Die Anfragen können nicht in der Abwesenheit des Ziel-Objekts gültig bleiben.	Die Ergebnisse können nicht in der Abwesenheit des Ziel-Objekts gültig bleiben.	keine
TII	Zusätzlich zur Nachricht sind noch weitere Informationen enthalten	Die Anfrage kann beim Client-Router dauerhaft gespeichert und ist somit unabhängig von der Erreichbarkeit des Ziel-Objekts.	Die Ergebnisse können beim Client-Router dauerhaft gespeichert und ist somit unabhängig von der Erreichbarkeit des Client-Objekts.	Client-Router, Ziel-Router.

3 Ein Anwendungsbeispiel des TII

In diesem Abschnitt wird ein Anwendungsbeispiel des TII vorgestellt. Daran wird der Ablauf des TII Nachrichtenaustauschs gezeigt.

Schritt 1: Anfrage initiieren Der Anwender hat eine Anwendung auf dem Laptop gestartet. Er führt eine Aktion durch, die zu einem entferntem Aufruf führt. Das Laptop ist momentan nicht an das Festnetz angeschlossen. Die Anwendung nutzt den ORB, um die Anfrage abzuschicken. Der ORB versucht zunächst, mit Hilf des normalen AMI die Anfrage an das Ziel-Objekt weiterzuleiten. Da dies nicht gelingt, schickt er die Anfrage an den lokalen Client-Router, der diese Abfrage zwischenspeichert. Das Verfahren ist für den Client transparent. Siehe Abbildung 6.

Abbildung 6: Anwendungsszenario

Schritt 2: Router identifizieren Wie findet der Client-ORB einen Client-Router? Das hängt von der Implementierung des ORBs ab. Es gibt verschiedene Möglichkeiten. Da ein CORBA-Router das IDL-Interface Messaging (CORBA Messaging Spezifikation) implementiert, hat der CORBA-Router eine Objektreferenz wie ein normales CORBA-Objekt. Deshalb kann der Client-ORB beispielsweise in

der Interoperable Objektreferenz (IOR) des Ziel-Objekts, die in der Anfrage enthalten ist, nachschauen, um die Information über das Routing zu bekommen. Zum anderen kann der Client-ORB so eingestellt werden, so daß der lokale Router verwendet wird. Bei diesem Beispiel verwendet diese Anwendung den lokalen Client-Router wie Abbildung 7 gezeigt. Abbildung 8 zeigt ein Beispiel, bei dem ein gemeinsamer Client-Router in einer LAN- Umgebung eingesetzt wird.

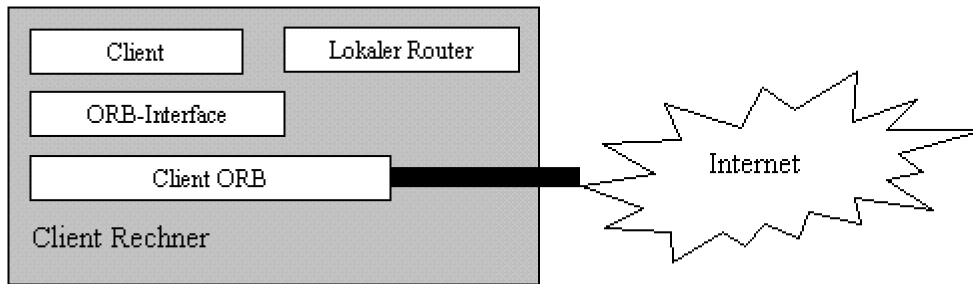


Abbildung 7: lokaler Client Router

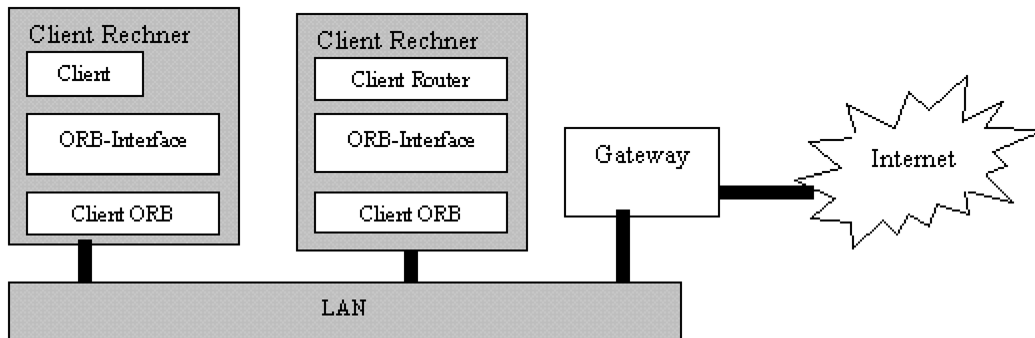


Abbildung 8: Client-Router im LAN

Schritt 3: Anfrage senden Der Client-ORB hat einen Client-Router gefunden. Er sendet die Anfrage asynchron an den Client-Router. Es gibt zwei Möglichkeiten, die davon abhängen, ob das Callback Modell oder das Polling Modell verwendet wird. Das Senden in beiden Modi wird im folgenden beschrieben.

Callback Modell

Falls die Client-Anwendung den ReplyHandler des Callback Modells verwendet, muß der Client-ORB das Interface Router des MessageRouting Moduls verwenden, um die Anfrage zu senden. Der Modul MessageRouting ist wie folgt definiert:

```
Module MessageRouting
{  enum ReplyDisposition { TYPED, UNTYPED };
    struct ReplyDestination
    { ReplyDisposition handler_type;
       Messaging::ReplyHandler  handler;
       sequence<string> typed_except_holder_repids;
    }

 struct RequestInfo
```

```
    {  RouterList visited;
       RouterList to_visit;
       Object target;
       Unsigned short profil_index;
       ReplyDestination reply_dest;
       Messaging::PolicyValueSeq selected_qos;
       RequestMessage payload;
    };
  typdef sequence<RequestInfo> RequestInfoSeq;
    interface Router
    { void send_request(in RequestInfo reg);
       void send_multiple_requests(in RequestInfo reg);
    }
}
```

Die Struktur RequestInfo enthält die Informationen über Router und Ziel-Objekt. Der Client-ORB kann diese Information verwenden, um den nächsten Router zu finden oder das Ziel-Objekt zu finden. Im Feld Payload ist die Anfrage eingepackt. Das Feld Handler hat den Typ von Messaging::ReplyHandler und enthält eine Objektreferenz auf den ReplyHandler der Cleint-Anwendung. Dieser ReplyHandler empfängt das Ergebnis. Der ORB ruft die Methode `send_request()` auf, um die Nachricht abzusenden.

Polling Modell

Der Client-ORB nutzt das Interface PersistentRequestRouter, um die Anfrage zu senden, falls die Client-Anwendung ein Polling-Objekt verwendet.

```
Module MessageRouting
{
struct RequestInfo { ..... gleich wie oben };
interface PersistentRequest;
{  readonly attribute boolean reply_availble;
   GIOP::ReplyStatus get_reply
    (in boolean blocking,
     in unsigned long timeout,
     out MessageBody reply_body) raises (ReplyNotAvailable);

    attribute Messaging::ReplyHandler associated_handler;
}

interface PersistentRequestRouter
{ PersistentRequest create_persistent_request
   ( in unsigned short profile_index,
     in Routerlist to_visit;
     in Object target,
     in CORBA::PolicyList current:qos,
     in RequestMessage payload );
}
```

Der Client-ORB ruft die Methode `create_persistent_request` aus dem Interface PersistentRequestRouter des MessageRouting Moduls auf und erhält eine Objektreferenz auf PersistentRequest, das im Router implementiert wird. Die benötigten Informationen über Router und das aufgerufene Ziel-Objekt werden beim Aufruf von `create_persistent_request` als Parameter an `create_persistent_request` übergeben. Mit Hilfe der Objektreferenz auf PersistentRequest kann der Client einerseits ermitteln, ob das Ergebnis bereits eingetroffen ist und andererseits das Ergebnis abholen. Das Attribute `reply_available` zeigt, ob das Ergebnis bereits eingetroffen ist. Mittels `get_reply` kann das Ergebnis abgeholt werden.

Schritt 4: Ziel-Router aufrufen Der Client-Router schickt die Anfragen an einen weiteren Client-Router oder einen Ziel-Router. Ein Ziel-Router schickt diese Anfragen nicht mehr an weitere Router, sondern verteilt die Anfragen auf die verschiedenen Ziel-Objekte. Der Ziel-Router führt hierfür die synchrone Methodenaufrufe auf den Ziel-Objekten aus und empfängt anschließend die Ergebnisse. Beim TII Modell müssen die Ziel-Objekte nicht geändert werden.

Schritt 5: Ergebnis zurückschicken Wenn das Ziel-Objekt den Rückgabewert der Methode als Ergebnis an den Ziel-Router zurückschickt, wandelt der Ziel-Router das Ergebnis in eine Anfrage um. Der Ziel-Router schickt das Ergebnis an den Client-Router. Der Client-Router bearbeitet den Zustand von `reply_available` (bei Polling Modell). Der Client kann diesen Zustand überprüfen und das Ergebnis abholen. Sobald die Anwendung wieder gestartet wird, sind die Ergebnisse verfügbar.

4 Vorteile des Asynchronen Nachrichtenaustauschs

In diesem Abschnitt werden die Vorteile des AMI und des TII zusammengefaßt.

- Mit Hilfe des Callback- und Polling Modells des normalen AMI kann der Client mehrere bidirektionale Aufrufe ohne zusätzlichen Thread verwalten, da die Probleme des Thread-Blockierung nicht mehr existieren. Es ist nicht notwendig, den Asynchronen Nachrichtenaustausch mit Hilfe des DII zu implementieren. Auf dieser Weise wird die Implementierung der Client Anwendungen vereinfacht.

- Die Einführung des AMI hat keinen Einfluß auf die Ziel-Objekte. Die Implementierung der Ziel-Objekte muß nicht verändert werden, da AMI nur die Client-Seite betrifft.

- Das TII Modell benutzt Store und Forward. Die Ziel-Objekte im TII Modell müssen nicht geändert werden, da Ziel-Router eingesetzt werden. Die Ziel-Router rufen synchron die Methode des Ziel-Objekts auf.

- Mehr Flexibilität. Bei dem TII Modell ist es möglich, das Ergebnis nicht nur durch den initial aufrufenden Client abzuholen, sondern auch durch einen anderen Client.

- Die Einführung von AMI in CORBA wird Anwendungen auf mobilen Geräten mehr Möglichkeiten zur Verfügung stellen. Mobile Geräte können aus Kostengründen nicht ununterbrochen sein. Das Store and Forward von Nachrichten ist sinnvoll für dieses Anwendungsumfeld. Die Verbindung wird intensiv und somit auch sparsam ausgenutzt.

5 Dienstqualität in AMI

In diesem Abschnitt wird beschrieben, wie man Dienstqualität (Quality of Service - QoS) definieren kann. In der neuen Spezifikation von CORBA 3.0 wird ein QoS Framework eingeführt. Das QoS Framework unterstützt den asynchronen und synchronen Nachrichtenaustausch bzw. asynchrone und synchrone Methodenaufrufe. Innerhalb dieses Frameworks werden alle Eigenschaften als Schnittstelle definiert, die von CORBA:Policy abgeleitet wird. Das QoS Framework ermöglicht der Anwendung die Definition von QoS in verschiedenen Schichten auf der Clientseite. Andererseits gibt es auch eine Schnittstelle für QoS auf der Serverseite.

5.1 Client-Seite

Mittels des neuen QoS Frameworks gibt es drei Schichten auf der Clientseite, auf denen man QoS definieren kann. Abbildung 9 zeigt die drei Schichten.

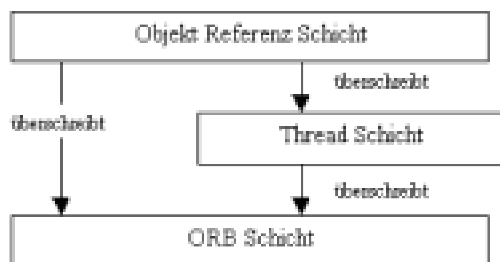


Abbildung 9: Architektur von QoS auf Client Seite

- Thread
  Eine Client-Anwendung enthält einen oder mehreren Threads. Jeder Thread ruft verschiedene Methoden von Ziel-Objekten auf. Man kann für jeden Thread spezifische QoS definieren. Die QoS gilt nur für die Methodenaufrufe aus diesem Thread. Außerdem kann die QoS-Policy eines Threads, die QoS-Definition im ORB überschreiben.

  Die Definition von QoS auf der ORB Schicht gelten für alle Anfragen, die dieser ORB absendet.

- Objektreferenz
  In der Client-Anwendung wird eine Referenz auf das Ziel-Objekten verwendet, um eine Methode des Ziel-Objekts aufzurufen. Innerhalb eines Threads können mehrere Objektreferenzen existieren. Man kann auch die QoS für eine bestimmte

Objektreferenz definieren. Die auf eine Objektreferenz basierende QoS-Definition überschreibt die Definition im Thread und im ORB.

5.2 Server-Seite

CORBA spezifiziert einige Schnittstellen für QoS auf der Server-Seite. Die Definition für QoS auf der Server-Seite wird im POA (Portable Object Adapter) durchgeführt. Die QoS wird beim Erzeugen des POAs festgelegt. Alle Objekte bzw. ihre Objektreferenz, die vom POA erzeugt werden, erhalten automatisch die QoS-Definition des POAs. Hat man beispielsweise die Policy TransactionPolicy für den POA definiert und der POA erzeugt ein Objekt, so fordert die Policy an, daß der Methodenaufruf auf das Objekt nur innerhalb einer Transaktion vorgenommen wird. Wenn der Client ORB eine Anfrage, die sich auf eine Methode des Objekts bezieht, sich aber außerhalb einer Transaktion befindet, erhält, wird diese Anfrage verweigert.

6 Beispiel für Programmierung mit AMI

In diesem Abschnitt wird ein C++ Programm, das das Callback Modell des AMI benutzt, vorgestellt. Wie Abbildung 10 zeigt gliedert die Implementierung des Callback Modells im AMI im wesentlichen in vier Schritte.

Schritt 1: Eine Schnittstelle Quoter wird in IDL definiert. Sie wird als Ziel-Objekt wie üblich vom Server-Programmierer implementiert.

Schritt 2: Diese Schnittstelle Quoter wird durch einen Compiler, der das Callback Modell des AMI unterstützt, kompiliert. Dabei werden zwei vorläufigen Implied-IDL, ein Implied-IDL für Client und ein Implied-IDL für Callback erzeugt. Die Implied IDL bedeutet, daß der IDL Compiler zuerst einige zusätzliche Informationen zur Standard IDL Definition einfügt und dann die ursprüngliche IDL-Definition und Implied IDL in die Stubs kompiliert. Das Implied IDL für Callback wird in Schritt 3 abgeleitet. Die Benennungskonvention der Funktionen und der Attribute im Implied IDL ist, daß das `sendc_` vor den ursprünglichen Namen der Funktionen und Attribute eingefügt wird. Der erste Parameter der Funktion `sendc_xxx` ist immer eine Objektreferenz auf Callback vom Typ `AMI_xxxHandler_ptr`, wobei xxx der Namespace ist. `AMI_xxxHandler_ptr` wird automatisch beim Client-ORB registriert, wenn der Client sie verwendet.

Schritt 3: Für das Callback Modell muß der Client-Programmierer eine Klasse vom Interface `AMI_QuoterHandler` im Namespace `POA_Stock`, ableiten und diese Klasse implementieren. In diesem Beispiel erhält diese Klasse den Namen `My_Aync_Stock_Handler`. Die Objektreferenz auf das Objekt der Klasse `AMI_QuoterHandler` wird beim Aufruf des `sendc_get_quote()` benötigt.

Schritt 4: Der Client wird implementiert. Zunächst werden die benötigten Objekte, die in Schritte 1-3 definiert wurden, initialisiert. Anschließend wird der Aufruf durchgeführt. Diese Funktion wird nicht gesperrt, nachdem der Aufruf ausgeführt wurde. Sobald der Client-ORB das Ergebnis erhält, wird

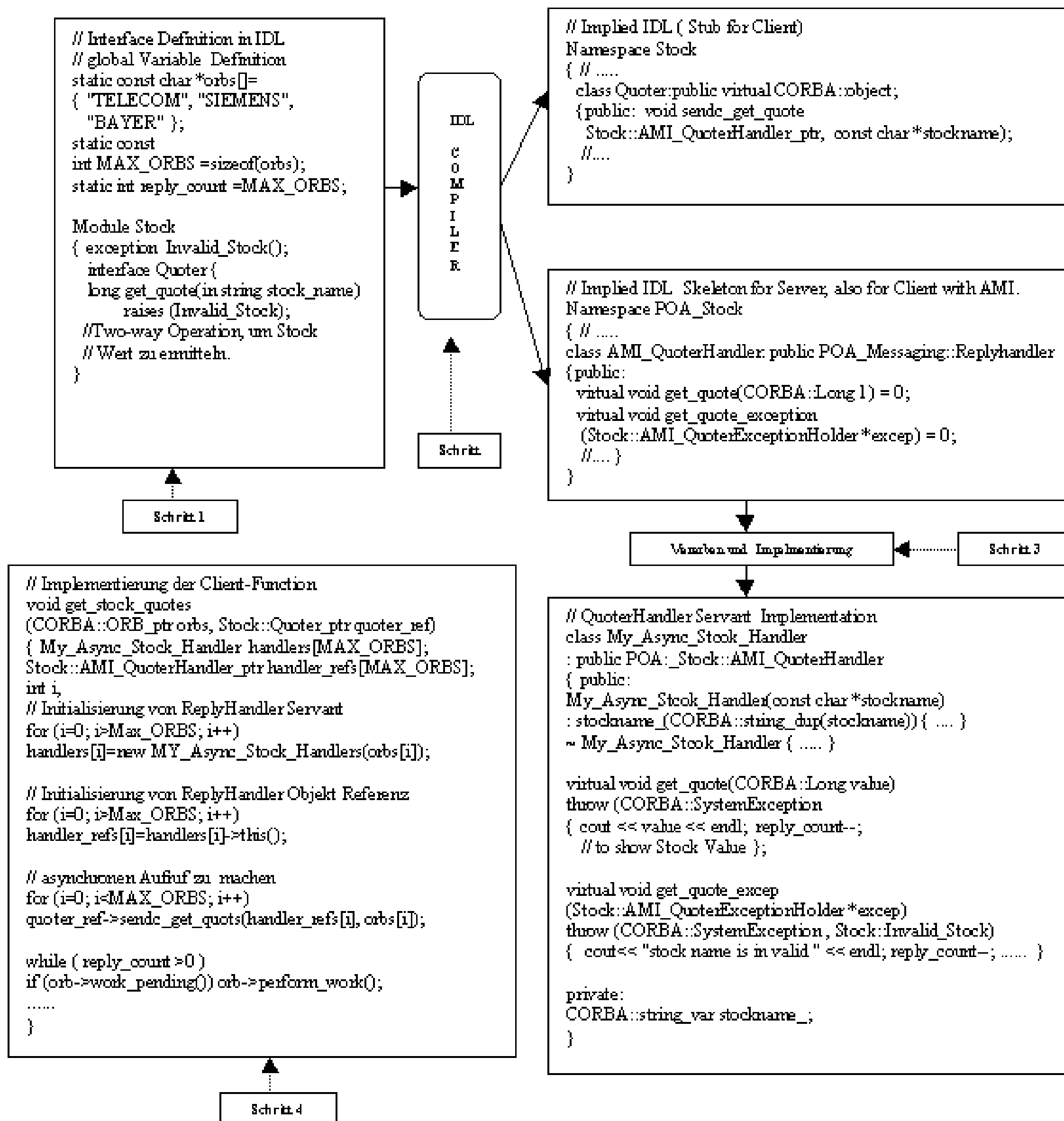


Abbildung 10: Implementierungsbeispiel

die Funktion `orb->work_pending()` den Wert Wahr haben. Die Methode `get_quote()` von `My_Aync_Stock_Handler` wird automatisch durch den Aufruf von `orb->perform_work()` ausgeführt. Der Serverobjekt soll wie üblich implementiert werden, da AMI die Implementierung des Serverobjekts nicht betrifft.

7 Fazit

In diesem Beitrag werden die wesentliche Eigenschaften und Funktionen des AMI beschrieben. Das Callback Modell, das Polling Modell vereinfachen die Entwicklung der Programme, die Nebenläufigkeit benötigen. TII bietet neue Möglichkeiten für Client-Programme an, die eine sichere Zustellung der Anfrage anfordern. Diese neue Leistungsfähigkeit wird mit vorhandenen Nachrichtenaustauschsmechanismen eine verbesserte verteilte Umgebung schaffen.

Literatur

[D. C98a] S. Vinoski D. C. Schmidt. An Introduction to CORBA Messaging. *Object Interconnections* Band 15, 1998.

[D. C98b] S. Vinoski D. C. Schmidt. Programming Asynchronous Method Invocation with CORBA Messaging. *Object Interconnections* Band 16, 1998.

[D. C98c] S. Vinoski D. C. Schmidt. Time Independent Invocation and Interoperable Routing. *Object Interconnections* Band 17, 1998.

[Robe97a] Dan Harkey Robert Orfali. *Client/Server Programming with Java and CORBA*. John Wiley & Sons. 1997.

[Robe97b] Jeri Edwards Robert Orfali, Dan Harkey. *Instant CORBA*. 1997.

Abbildungsverzeichnis

1 Der synchroner bidirektionale Nachrichtenaustausch 16

2 Klassifikation des CORBA Nachrichtenaustauschs 17

3 Architektur des Polling Modells 18

4 Architektur des Callback Modells 18

5 Architektur und Ablauf des TII 20

6 Anwendungsszenario 22

7 lokaler Client Router 23

8 Client-Router im LAN 23

9 Architektur von QoS auf Client Seite 26

10 Implementierungsbeispiel 28

Sicherheitsmechanismen in CORBA

Olaf Titz

Kurzfassung

In verteilten Anwendungen ist ein erhöhter Aufwand zur Einhaltung der Sicherheitsanforderungen erforderlich. Anwendungsunterstützung für verteilte Systeme erfordert sowohl anwendungstransparente als auch anwendungsspezifische Sicherheitsmechanismen. CORBA stellt eine Vielzahl solcher Mechanismen zur Verfügung. Diese Arbeit gibt einen Überblick darüber. Dabei werden insbesondere das Prinzip der Authentikation unter CORBA, die Delegation von Sicherheitsattributen und die verwendeten unterliegenden Netzwerkprotokolle genauer beschrieben.

1 Einleitung

Verteilte Systeme erfordern gegenüber monolithischen Anwendungen sowohl direkt erhöhte Anforderungen an die Sicherheit als auch zusätzlichen Aufwand bei der Modellierung und Programmerstellung. Ersteres folgt aus der Kommunikation zwischen Komponenten über Netzwerke, die potentielle Angriffswege öffnet, letzteres ergibt sich aus der Trennung von Anwendungen in Komponenten, zwischen denen implizite und explizite Vertrauensbeziehungen bestehen.

1.1 Sicherheitsanforderungen in verteilten Systemen

Die Kommunikation zwischen Komponenten einer verteilten Anwendung erfordert die Einhaltung grundlegender Sicherheitskriterien:

- *Integrität*: Daten dürfen bei der Übertragung nicht unerkennbar verfälscht oder verändert werden;
- *Vertraulichkeit*: Daten müssen gegen Kenntnisnahme durch Unberechtigte geschützt werden;
- *Authentikation*: Jede an einem Vorgang beteiligte Instanz muss ihre Identität und Berechtigung nachweisen;
- *Unwiderrufbarkeit*: Die Ausführung eines Vorgangs muss auch im Nachhinein nachweisbar bleiben.

Die Einhaltung dieser Anforderungen, insbesondere der Schutz vor Angriffen, aber auch vor unabsichtlicher Verletzung der Sicherheit durch Implementations- oder Bedienungsfehler, muss durch das System sichergestellt werden. Dazu bedient es sich als grundlegender Mechanismen der Verfahren der Kryptographie [Schn96].

Die Gesamtheit der Systemdienste, die für die Einhaltung der Sicherheit verantwortlich sind und daher gegenüber den Anwendungen vertrauenswürdig sein müssen, wird als *Trusted Computing Base* (TCB) bezeichnet. In einem CORBA-System umfasst sie die ORBs, die Kommunikationswege zwischen ORBs und die unterliegenden sicherheitsrelevanten Systemkomponenten.

1.2 Anwendungsunterstützung für Sicherheitsdienste

Zur Einhaltung der Sicherheitsanforderungen sind eine Anzahl unterschiedlicher Sicherheitsmechanismen im Gebrauch, die sich in *anwendungstransparente* und *anwendungsunterstützende* Mechanismen unterteilen lassen.

1.2.1 Anwendungstransparente Sicherheitsdienste

Anwendungstransparente Sicherheitsmechanismen arbeiten in der Kommunikation zwischen Komponenten und sichern diese ab, ohne dass die Komponenten dafür besonders eingerichtet werden müssen. Ein typisches Beispiel ist die Verwendung des SSL-Protokolls (*Secure Sockets Layer*) [FrKK96] als Transportschicht. Dabei wird ein beliebiger Datenstrom verschlüsselt, signiert und optional die Identität der Kommunikationspartner verifiziert. Es ist nun möglich, beliebige Anwendungsprotokolle über diesen gesicherten Datenstrom zu betreiben. Insbesondere können auch Anwendungen, die selbst keine besonderen Sicherheitsmechanismen enthalten (*Security Unaware Applications*), auf diese Weise mit höherer Sicherheit betrieben werden. Der ursprüngliche Haupteinsatzzweck von SSL für das nur schwach gesicherte HTTP-Protokoll ist ein Beispiel dafür. Der Betrieb des CORBA-IIOP über SSL wird unter 6.3 weiter erläutert.

1.2.2 Anwendungsunterstützende Sicherheitsdienste

Viele Anwendungen erfordern spezifische Sicherheitsmechanismen, um eigene, in der Anwendung festgelegte Sicherheitsanforderungen zu erfüllen (*Security Aware Applications*). Hierzu müssen unterstützende Sicherheitsdienste verfügbar sein, die von der Anwendung explizit benutzt werden, also spezifische Programmierung erfordern. Der CORBA Security Service enthält eine Anzahl solcher Dienste, die vom ORB zur Verfügung gestellt werden. Es ist zu beachten, dass die Implementation der meisten dieser Dienste nicht zwingend vorgeschrieben ist.

Beispiele für Anwendungsunterstützung sind Dienste zur Handhabung von Identitäten und Berechtigungen, wie sie CORBA in Form der Credentials (siehe 3) bereitstellt. Sie erlauben es einer Anwendung, gegenüber bestimmten Komponenten in einem anwendungsdefinierten Sicherheitskontext aufzutreten. Insbesondere wird hierdurch die Einhaltung des „Prinzips des geringsten Privilegs“ ermöglicht, nach dem kein Vorgang mit Berechtigungen ablaufen soll, die für diesen Vorgang nicht benötigt werden.

2 Der CORBA Security Service

Der Standard für den CORBA Security Service umfasst eine große Zahl an Objekten und Schnittstellen. Abbildung 1 zeigt eine Zusammenfassung davon nach Funktions- und Implementationsaspekten geordnet.

Funktionell deckt der Security Service folgende Bereiche ab:

- *Authentication and Security Association*
  - auf Anwendungsebene: Erlangung und Handhabung von Identitäten und Berechtigungen (siehe 3),
  - auf Implementationsebene: Bereitstellung von Mechanismen dazu;
- *Authorization and Access Control*
  - auf Anwendungsebene: Festlegung von Rechten und Bedingungen zum Zugriff auf Dienste,
  - auf Implementationsebene: Durchsetzung der Zugriffskontrolle;
- *Accountability*
  - auf Anwendungsebene: Anforderungen von Beweisen und Protokollen über Vorgänge,
  - auf Implementationsebene: zuverlässige und unanfechtbare Erstellung solcher Protokolle sowohl für die Anwendungen (vgl. 5) als auch für die Administration (*Auditing*).
- Hinzu kommt ein alle Bereiche überdeckendes *Policy Management* zur Definition und Administration der Sicherheitsregeln (siehe 4), sowie die Implementation von Sicherheitsdiensten auf unterer Ebene (6).

Nicht alle Bestandteile des CORBA Security Service müssen zwingend von jedem ORB implementiert sein. Der Standard definiert Konformitätsklassen, die jeweils nur einen Teil der Sicherheitsdienste bereitstellen:

- Alle Sicherheitsfunktionen und -objekte sind in zwei Klassen eingeteilt:
  - Level 1 – grundlegende Funktionalität für „security unaware applications“ und Anwendungen, die nur eingeschränkte Anforderungen stellen;
  - Level 2 – weitergehende Funktionalität, Policy-Administration.
- Die Unwiderrufbarkeitsdienste gehören nicht zum Kernstandard und sind ausdrücklich als optional bezeichnet.
- ORBs können in einer Weise implementiert werden, dass ihre Sicherheitsdienste nicht eingebaut, aber durch externe Komponenten nachrüstbar sind. Solche ORBs heißen „security-ready“ (*nicht* „secure“).

Anwendungen, die auf Sicherheitsdienste zugreifen, müssen prüfen, ob der jeweilige Dienst überhaupt vorhanden ist.

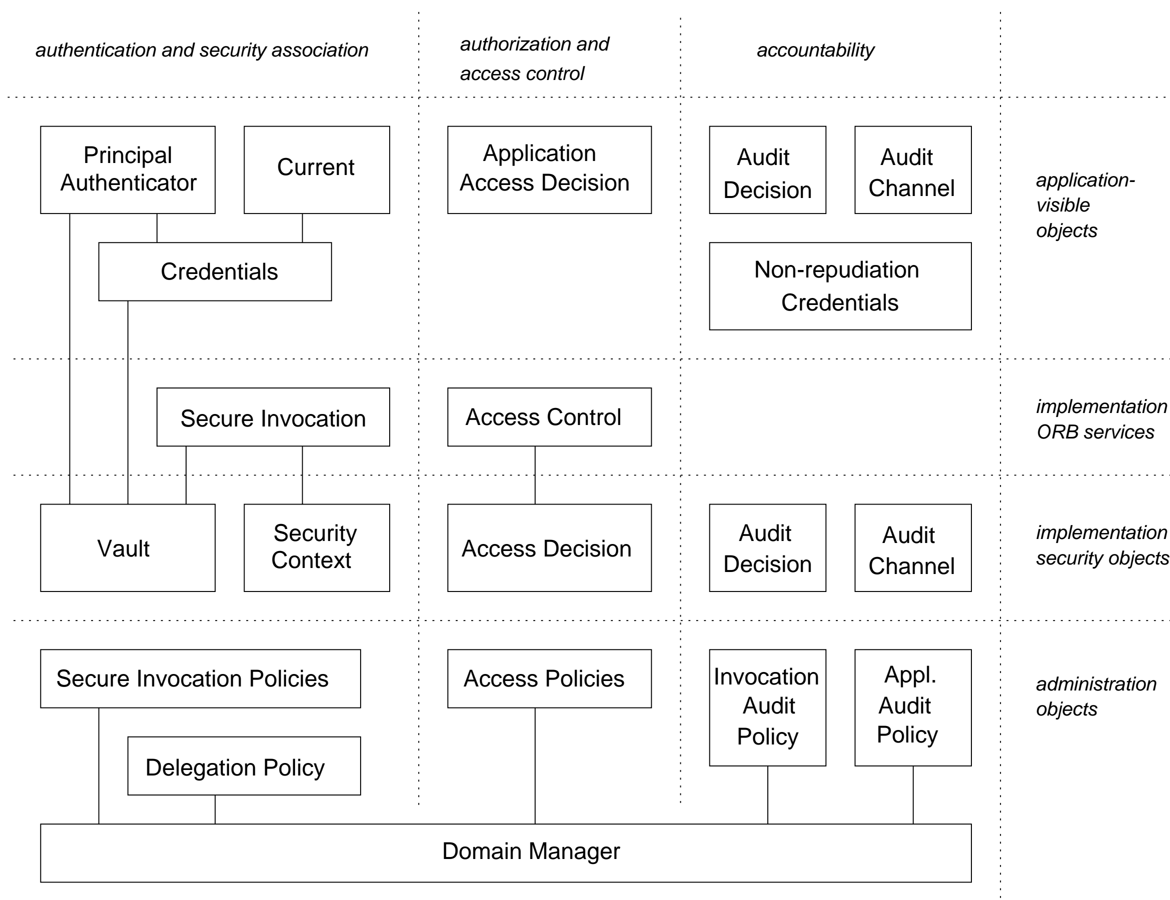


Abbildung 1: CORBA Security Objekte (aus [OMG98])

3 Authentikation und Credentials

Von zentraler Bedeutung ist die Authentikation und der Nachweis von Berechtigungen zur Nutzung bestimmter Dienste. In großen verteilten Systemen können komplexe Beziehungen entstehen, welcher Dienst von welchem Aufrufer unter welchen Bedingungen benutzt werden darf. Um die Verwaltung dieser Rechte möglichst universell zu halten, wurde die Authentikation in CORBA anhand folgender allgemeiner Begriffe, die gleichzeitig als Objekte im System existieren können, definiert:

- *Principal* ist ein Nutzer, Dienst, Objekt oder eine sonstige identifizierbare Komponente.
- *Identity* ist die Identifikation eines Principals, unter der er bei Vorgängen in Erscheinung tritt. Ein Principal kann mehrere Identities besitzen.
- *Privilege* ist eine Berechtigung, einen Dienst zu nutzen oder sonst einen Vorgang zu initiieren. Es beschreibt genau die Bedingungen, unter denen die Benutzung erfolgen darf.
- *Security Attributes* sind Identity, Privilege und eventuelle weitere sicherheitsrelevante Informationen. Attribute können voneinander abhängen. Sie können an andere Principals weitergegeben werden (Delegation, siehe 3.1).

- *Authentication* ist der Vorgang, mit dem ein Principal Attribute erlangen kann, indem er seine Berechtigung dazu in geeigneter Weise nachweist.

- *Credentials* sind Mengen von Security Attributes.

Ein neu in ein System kommender Principal (z. B. ein Benutzer beim Login-Vorgang, aber auch ein neu gestarteter Serverdienst) muss in der Regel als erste Aktion durch Authentikation einen Satz Sicherheitsattribute erlangen. Hierzu können beliebige Methoden der Authentikation verwendet werden. Das Ergebnis ist mindestens eine Identity und eventuell sonstige Attribute. Ein Sonderfall sind *Public*-Attribute, die ohne Authentikation, also anonym und von jedem Principal, erlangt werden können.

Bei einem Methodenaufruf über den ORB liefert der Aufrufer die notwendigen Sicherheitsattribute als *Credentials*-Objekt mit. Sowohl das Zielobjekt als auch die an dem Aufruf beteiligten Komponenten (ORB und Netzwerk) können anhand dieser Attribute über die Berechtigung entscheiden, den Zugriff für Administration oder Unwiderrufbarkeitsdienste aufzeichnen oder sonstige Auswertungen vornehmen. Die genaue Festlegung dieser Bedingungen ist Aufgabe der *Security Policy*, die durch Administration festgelegt wird (siehe 4) und deren Einhaltung von der TCB garantiert werden muss.

3.1 Delegation

In einem verteilten System können Komponenten, die als Server von einem Aufrufer aktiviert werden, ihrerseits als Aufrufer andere Komponenten aktivieren. Dieser Vorgang kann auch ohne Wissen des ursprünglichen Aufrufers erfolgen. Dabei muss besonders beachtet werden, welche Sicherheitsattribute in dem indirekten (delegierten) Aufruf wirksam sind. Der CORBA Security Service sieht hierfür mehrere Schemata vor, die im folgenden erläutert werden. Welches davon im Einzelfall benutzt wird, ist sowohl von der Modellierung der Anwendung als auch von der Implementation (Verfügbarkeit) abhängig.

In jedem Fall kann ein Aufrufer spezifizieren, welche seiner Sicherheitsattribute vom unmittelbaren Ziel benutzt werden und welche über weitere Aufrufe delegiert werden dürfen. Ein Aufrufziel kann bestimmen, welche Sicherheitsattribute und welche Herkunft es akzeptiert und welche es delegiert. Aus der Kombination ergibt sich das letztlich benutzte Schema, wobei die Zuordnung auch fallweise wechseln kann.

3.1.1 No Delegation

Die Sicherheitsattribute des Aufrufers werden nur vom unmittelbaren Ziel benutzt und nicht delegiert. Bei einem indirekten Aufruf wird stattdessen der Sicherheitskontext

des zwischenliegenden Objekts benutzt. Das indirekte Ziel erkennt nur den zwischenliegenden Aufrufer.

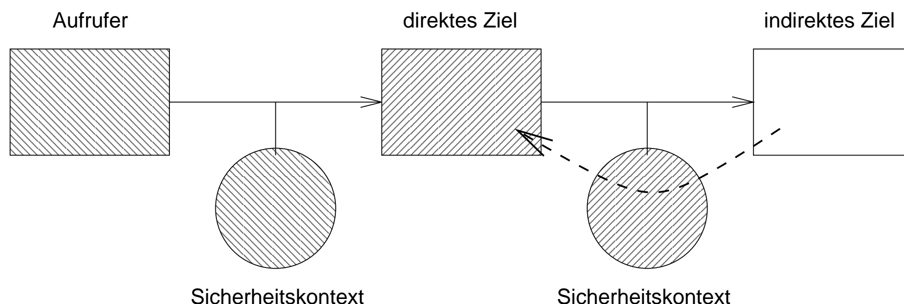


Abbildung 2: No Delegation

No Delegation erfordert vom Zielobjekt Vertrauen in den zwischenliegenden Aufrufer, die Sicherheitsattribute des ursprünglichen Aufrufers entsprechend einer gemeinsamen Policy zu handhaben.

3.1.2 Simple Delegation

Die Sicherheitsattribute des Aufrufers werden vom unmittelbaren Ziel unverändert delegiert. Ein indirekter Aufruf erfolgt im Sicherheitskontext des ursprünglichen Aufrufers.

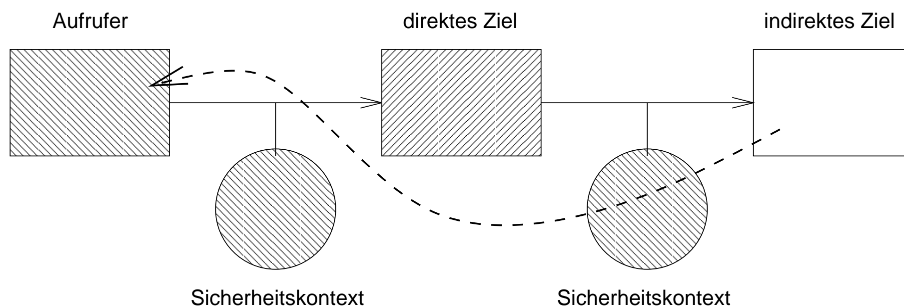


Abbildung 3: Simple Delegation

Bei Simple Delegation vertraut der ursprüngliche Aufrufer allen Zielobjekten. Eine Weiterdelegation durch ein nicht vertrauenswürdiges Ziel könnte Aktionen veranlassen, die der ursprüngliche Aufrufer nicht beabsichtigt. Dieses Aufrufmodell entspricht einem simplen Prozeduraufruf.

3.1.3 Composite Delegation

Die Sicherheitsattribute des Aufrufers werden vom unmittelbaren Ziel delegiert, die Sicherheitsattribute des unmittelbaren Ziels werden hinzugefügt. Ein indirekter Aufruf erfolgt in beiden durch ihre Identität bestimmbaren Sicherheitskontexten.

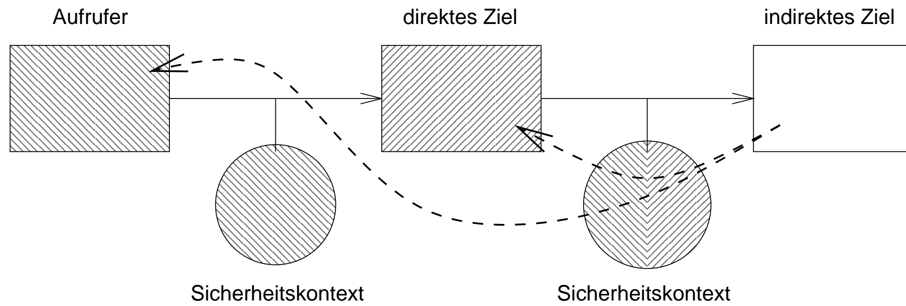


Abbildung 4: Composite Delegation

Je nach Mächtigkeit der delegierbaren Attribute (vgl. 3.2) erfordert dieses Modell gegenseitiges Vertrauen in der Weise der No Delegation wie auch der Simple Delegation. Ein vergleichbares Modell ist der Aufruf von privilegierten oder *set-uid*-Prozessen in Betriebssystemen.

3.1.4 Combined Privileges Delegation

Die Sicherheitsattribute, ausgenommen die Identität, des Aufrufers werden vom unmittelbaren Ziel seinem eigenen Sicherheitskontext hinzugefügt. Ein indirekter Aufruf erfolgt nur noch in einem Sicherheitskontext, der die Attribute beider Objekte enthält. Im Unterschied zur Composite Delegation kann ein indirektes Ziel die Herkunft der Sicherheitsattribute nicht bestimmen.

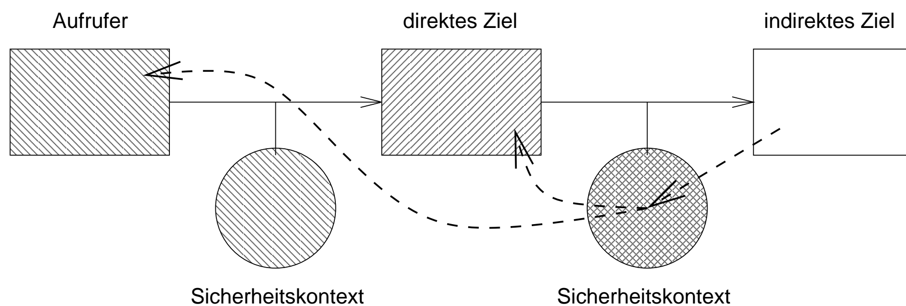


Abbildung 5: Combined Privileges Delegation

Hinsichtlich der Vertrauensbeziehung gilt Ähnliches wie bei der Composite Delegation.

3.1.5 Traced Delegation

Die Sicherheitsattribute des Aufrufers werden vom unmittelbaren Ziel delegiert, diejenigen des unmittelbaren Ziels werden als Kette hinzugefügt. Das Zielobjekt kann die Aufrufreihenfolge zurückverfolgen.

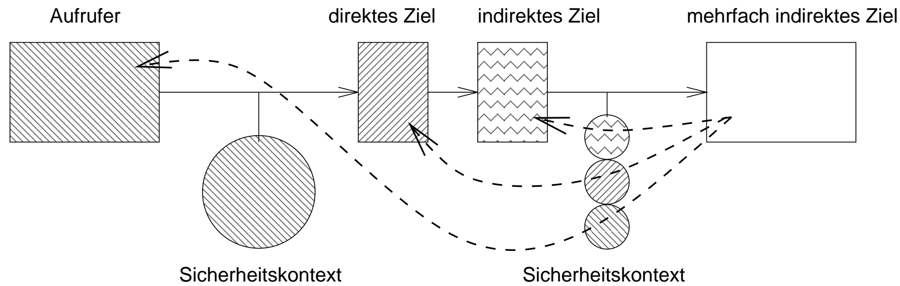


Abbildung 6: Traced Delegation

Aus Sicht des Aufrufers besteht meist kein funktionaler Unterschied zwischen Composite, Combined und Traced Delegation. Im Ergebnis kann das Zielobjekt die Sicherheitsattribute aller beteiligten direkten und indirekten Aufrufer verwenden. Ein Unterschied ergibt sich jedoch im Zielobjekt, das unter Umständen je nach Anwendung nur ein bestimmtes Aufrufschema zulässt.

3.2 Interoperabilitätsklassen

Die Verfügbarkeit der Delegationsschemas ist implementationsabhängig. Der Standard definiert drei Klassen (*Common Security Interoperability Levels*, CSI), in denen die Unterstützung für den Transfer von Sicherheitsattributen durch einen ORB bereitgestellt werden kann:

- CSI Level 0 kennt nur die Identität des Aufrufers als Sicherheitsattribut und keine Delegation. Alle indirekten Aufrufe erfolgen somit nach dem Schema „No Delegation".
- CSI Level 1 kennt nur die Identität des Aufrufers als Sicherheitsattribut, welche nur uneingeschränkt delegiert werden kann und erlaubt damit „No Delegation" und „Simple Delegation".
- CSI Level 2 umfasst die volle Funktionalität von Sicherheitsattributen: Identität, Privilegien und Berechtigungen, einschließlich solchen Attributen, die die Delegation von Privilegien einschränken. Damit sind alle Schemata möglich.

4 Security Policy

Security Policy ist die Gesamtheit der Sicherheitsregeln, die für eine bestimmte, administrativ verbundene Menge von Objekten (*Security Policy Domain* oder kurz *Security Domain*) gelten. Sie umfasst Authentikationsanforderungen, Zugriffskontrolle,

Anforderungen über das Sicherheitsniveau verwendeter Objekte und Infrastruktur, Anforderungen an die Aufzeichnung von Aktionen (*Auditing*) und vieles mehr. Nahezu alle Bestandteile des CORBA Security Service verfügen über Schnittstellen zum Policy-Management, mit dem die Anforderungen der Policy global und im Einzelfall festgelegt werden können. Die Einhaltung der Policy muss in sämtlichen betroffenen Diensten durch die Trusted Computing Base gewährleistet werden.

Ein großes verteiltes System kann mehrere Security Policy Domains umfassen. Dies ist immer dann der Fall, wenn hierarchisch strukturierte Teildomains vorliegen, aber auch, wenn sich Domains (zumeist nur in bestimmten Teilbereichen der Policy) überlappen oder Objekte aus verschiedenen Domains verbunden werden müssen. Dann wird eine Koordination der beteiligten Domains und die Auflösung von im Konflikt stehenden Anforderungen notwendig. Dies ist Aufgabe des (manuellen) *Security Policy Managements*.

Für den wichtigen Fall, dass (nicht nur einzelne) Objekte aus unterschiedlichen Domains aufeinander zugreifen müssen, wurde der Mechanismus der *Domain Federation* eingeführt. Dabei kann der Manager einer Domain festlegen, dass bestimmte Sicherheitsattribute aus einer anderen Domain in seiner Domain zu bestimmten Zwecken anerkannt werden.

5 Unwiderrufbarkeit

Zur Sicherstellung der Unwiderrufbarkeit von Vorgängen und der Kontrolle von Verantwortlichkeit für Vorgänge werden spezielle Dienste innerhalb einer Trusted Computing Base herangezogen, die sämtliche relevanten Daten in manipulationssicherer Weise aufzeichnen und beiden Seiten Nachweise über einen stattgefundenen Vorgang erbringen.

Diese Erstellung von Aufzeichnungen und Beweisen erfolgt in CORBA durch einen eigenen Dienst mittels *Non Repudiation Credentials*-Objekte, aus denen die *Non Repudiation Tokens*, die eigentlichen Beweise, nach Bedarf erzeugt werden. Die Verwaltung dieser Objekte obliegt den Anwendungen. Ein Dienst zur Langzeitspeicherung von Non Repudiation Tokens und ein Dienst zur Auflösung von Konflikten um die Gültigkeit der Tokens (*Adjucator*) wären notwendig, sind aber in CORBA derzeit nicht standardisiert. Ihre Anwendung bleibt daher auf Spezialfälle beschränkt.

6 Protokolle

CORBA spezifiziert eigene Sicherheitsprotokolle und erlaubt die Verwendung vorhandener Protokolle zu bestimmten Zwecken. Die Basismechanismen der Verschlüsselung, Signatur, Authentikation werden auf vorhandene Standards und Implementationen abgestützt.

6.1 Authentikation

Zur Authentikation der Benutzer und Instanzen und Erzeugung der initialen Sicherheitskontexte kann CORBA auf bestehende Authentikationsmechanismen zurückgrei-

fen. Insbesondere werden SPKM, Kerberos und CSI-ECMA unterstützt. Dazu dient das GSS-API, das eine standardisierte Programmierschnittstelle für Authentikationsdienste bereitstellt [Linn93, Linn96].

6.2 SECIOP

Zur sicheren Kommunikation zwischen ORBs wurde das Protokoll SECIOP definiert, welches das allgemeine Protokoll GIOP um Sicherheitsmechanismen ergänzt. SECIOP sitzt als zusätzliche Schicht zwischen GIOP und der Transportschicht und bildet eine sichere Einkapselung von GIOP-Nachrichten. (Abb. 7)

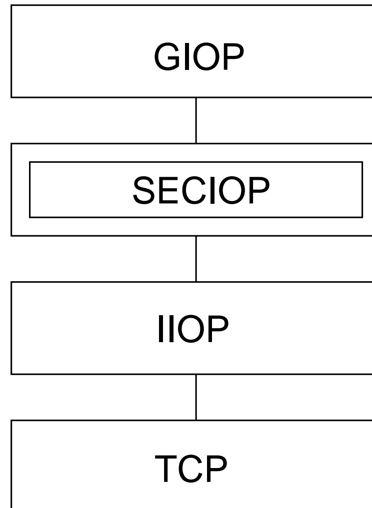


Abbildung 7: SECIOP

SECIOP handhabt eine beliebige Anzahl von Sicherheitsbeziehungen auf einer Verbindung der Transportschicht. Eine Sicherheitsbeziehung besteht aus einem Paar (eines je Seite) *Security Context*-Objekte. Anwendungsseitig stehen die Mechanismen zur Etablierung dieser Kontexte und ihrer Benutzung zum Transport von GIOP-Nachrichten zur Verfügung. Damit haben die Anwendungen die Kontrolle über die Sicherheitseigenschaften der Verbindungen, die zum Transport der Nutzdaten verwendet werden.

6.3 SSL

Eine oftmals wesentlich einfacher zu realisierende Alternative ist der Betrieb von GIOP über SSL (Abb. 8). SSL setzt auf einer TCP-Verbindung auf, deren Nutzdaten signiert und verschlüsselt übertragen werden. Zusätzlich ist im SSL-Standard optional vorgesehen, beim Verbindungsaufbau beide Seiten über X.509-Zertifikate zu authentisieren. Somit entsteht genau eine Sicherheitsbeziehung für eine TCP-Verbindung.

Der Authentikationsmechanismus muss beim Einsatz für CORBA verwendet werden, um eine verifizierbare Übertragung von Identitäten zu erreichen. Da diese Identität an die SSL-Schicht gebunden ist, lässt sich auf diese Weise nur CSI Level 0 (siehe 3.2) erreichen. Die Vorteile von SSL — Verschlüsselung der Verbindung, Authentikation

auf ORB-Ebene — kommen dafür auch „security unaware applications" zugute. Für die Anwendungen sind die Sicherheitseigenschaften der SSL-Verbindung im Gegensatz zu SECIOP im allgemeinen unsichtbar.

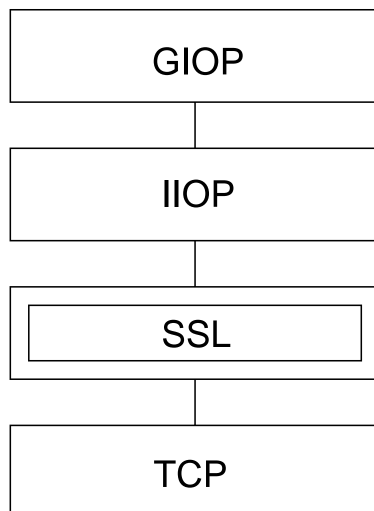


Abbildung 8: GIOP über SSL

6.4 Firewalls

Im Internet kommen CORBA-Anwendungen — auch aus Gründen der Abschottung von Netzwerken gegen Angriffe — oft über Firewalls hinweg zum Einsatz. Da solche Firewalls meist keine direkten Verbindungen zulassen, muss ein Proxy-Mechanismus verwendet werden. Welche Art von Proxy sich hierfür eignet, hängt von den Anforderungen der konkreten Anwendung ab. CORBA spezifiziert die Benutzung eines direkten TCP-Proxys, von SOCKS und eines speziellen CORBA-(GIOP)-Proxys.

6.4.1 TCP-Proxy

Unter einem TCP-Proxy versteht man einen einfachen Dienst, der TCP-Verbindungen auf der Firewall annimmt und an ein festgelegtes Ziel weiterleitet. Der Einsatz ist mit geringem Aufwand in nahezu allen Situationen möglich. Der Proxy selber hat im einfachsten Fall keinen Zugriff auf den Datenstrom, insbesondere kann er Zugriffskontrollen nur anhand der IP-Adressen durchführen. Solche Proxies können in beide Richtungen aufgesetzt werden. Sie sind für jede Anwendung eigens zu konfigurieren. Der aufrufende ORB muss anstelle der tatsächlichen TCP/IP-Adresse des Partners die Adresse des Proxys verwenden.

Als einfache Erweiterung liegt es nahe, im Proxy eine syntaktische Überprüfung des Datenstroms (auf korrektes GIOP) vorzunehmen, um Angriffe abzuwehren, die Reaktionen der ORBs auf absichtliche fehlerhafte Daten ausnutzen [One96]. Eine semantische Analyse der Daten würde dagegen eine CORBA-Umgebung im Proxy bedingen.

6.4.2 SOCKS

SOCKS [LGLK+96] ist ein Protokoll für den Aufbau beliebiger TCP-Verbindungen durch einen Client. Dabei öffnet der Client eine Verbindung zum Proxy und teilt diesem mit, zu welcher TCP/IP-Adresse die Verbindung hergestellt werden soll. Der Proxy baut die Verbindung zum Zielsystem auf und reicht sie an die bestehende Verbindung zum Client hin weiter.

Auch hier hat der Proxy keinen Zugriff auf die Nutzdaten und kann nur die Adressen auswerten. Ein einzelner Proxy kann eine große Anzahl von Verbindungen auf verschiedenen Protokollen handhaben, eine auch nur syntaktische Analyse der Nutzdaten ist ihm daher nicht möglich. Der Zugriff ist nur in eine Richtung möglich, ein Host kann nicht Client- und Serverfunktionalitäten über die Firewall gleichzeitig bereitstellen. SOCKS-Server gehören zur Standardausstattung jedes modernen Firewallsystems, daher verursacht eine SOCKS-basierte Lösung meist keinen zusätzlichen Aufwand. Der vorgeschaltete Dialog mit dem Proxy erfordert eine (einfache) Softwareanpassung des ORB, hierfür sind Standardbibliotheken vorhanden.

6.4.3 GIOP-Proxy

Ein GIOP-Proxy analysiert den Datenstrom und leitet die Verbindungen danach weiter. Ihm steht prinzipiell die volle Funktionalität von CORBA für Zugriffskontrolle und Verbindungsentscheidungen zur Verfügung. Insbesondere ist es möglich, bereits im Proxy Zugriffskontrollentscheidungen anhand der Sicherheitsparameter im Datenstrom zu treffen. Dadurch wird der Proxy flexibel und für höchste Sicherheitsanforderungen geeignet, jedoch auch sehr viel komplexer als die anderen Lösungen. Außerdem bietet die Analyse des Datenstroms einen Basisschutz gegen Angriffe durch bewusst falsche Daten.

6.4.4 SSL über Proxy

Beim Einsatz von SSL über Firewalls hinweg kann ein einfacher TCP-Proxy zum Einsatz kommen. Die Bedingungen hierfür entsprechen einem TCP-Proxy für rohes IIOP, wobei jedoch der Proxy selber keinerlei Zugriff auf den verschlüsselten Datenstrom haben kann.

Weitere Möglichkeiten ergeben sich aus einem SSL-zu-SSL-Proxy [Pint98], der den SSL-Sicherheitskontext umsetzt, oder einem TCP-zu-SSL-Proxy, der nur auf der unsicheren Seite der Firewall SSL verwendet. Dabei ist jeweils die der SSL-Verbindung innewohnende Authentikation zu beachten.

7 Praxis und Ausblick

Die CORBA Security Service Spezifikation krankt an der mit der hohen Leistungsfähigkeit zwangsläufig einhergehenden Komplexität. Um die Verfügbarkeit konformer Implementationen in absehbarer Zeit überhaupt zu ermöglichen, wurden verschiedene Ebenen der Standardkonformität festgelegt, die Implementation der meisten Dienste

ist optional. An vielen Stellen wird auf nicht näher spezifizierte unterliegende Dienste zurückgegriffen, die nicht überall in vollem Leistungsumfang bereitgestellt werden können. Dies gilt um so mehr, als rechtliche Einschränkungen die Verfügbarkeit der Implementationen vielerorts behindern.

Somit müssen viele existierende Systeme noch ohne den vollen Umfang des Security Service auskommen. Entsprechend selten sind „security aware applications“, die alle Möglichkeiten ausschöpfen. Die Entwicklung solcher Anwendungen wird zusätzlich dadurch erschwert, dass die korrekte Modellierung der notwendigen Sicherheitsattribute in dem Maße komplexer wird, wie das gegenseitige Vertrauen der einzelnen Komponenten ineinander sinkt (was die Notwendigkeit von security aware applications oft erst hervorruft).

Als einfachste Lösung, um CORBA-Anwendungen gegen netzbasierte Angriffe abzusichern, hat sich SSL in Kombination mit geeigneten Firewalls etabliert. Damit ist auch für „security unaware applications“ eine Möglichkeit verfügbar, einen wesentlichen Teil der Sicherheitsbedingungen weitgehend zu erfüllen.

In Zukunft muss sich noch zeigen, ob das Konzept des CORBA Security Service überhaupt sinnvoll im vollen Umfang zu verwirklichen ist. Unter Umständen könnten weniger leistungsfähige, aber auch weniger komplexe Teilmengen der Sicherheitsdienste leichter verfügbar und trotzdem technisch hinreichend sein.

Literatur

[FrKK96] Alan O. Freier, Philip Karlton und Paul C. Kocher. The SSL Protocol, Version 3.0. Protocol specification, Netscape Communications Corporation, Mountain View, Cal., 1996.

[KoNe93] John Kohl und B. Clifford Neuman. The Kerberos Network Authentication Service (V5). *RFC 1510*, September 1993.

[LaSc98] Ulrich Lang und Rudolf Schreiner. Schutz und Trutz. *iX*, Oktober 1998.

[LGLK+96] Marcus Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas und L. Jones. SOCKS Protocol Version 5. *RFC 1928*, März 1996.

[Linn93] John Linn. Generic Security Service Application Program Interface. *RFC 1508*, September 1993.

[Linn96] John Linn. The Kerberos Version 5 GSS-API Mechanism. *RFC 1964*, Juni 1996.

[OMG98] Object Management Group. *Security Service Specification*, 1998.

[One96] Aleph One. Smashing The Stack For Fun And Profit. *Phrack* Band 7, November 1996.

[Pint98] Erwin Pinter. Starkes SSL für kommerzielle Web-Clients – Eine beispielhafte Implementierung als Sicherheits-Proxy. Diplomarbeit, Universität Karlsruhe, 1998.

[Schn96] Bruce Schneier. *Applied Cryptography.* Wiley & Sons, New York. ISBN 0-471-12845-7, 2. Auflage, 1996.

Abbildungsverzeichnis

1 CORBA Security Objekte (aus [OMG98]) 34

7 SECIOP 40

8 GIOP über SSL 41

Transportprotokolle für drahtlose und mobile Endgeräte

Dennis Koslowski

Kurzfassung

Angesichts der gegenwärtigen Verbreitung mobiler Kommunikationsgeräte stellt sich das Problem deren Anbindung an das Internet. Für drahtlose Verbindungen sind niedrige Bandbreiten, hohe Fehlerraten, Halbduplex-Übertragung und häufige Verbindungsabbrüche die charakteristischen Eigenschaften. Diese Eigenschaften führen zu einer sehr geringen Leistungsfähigkeit von TCP auf drahtlosen Verbindungen. Um die Leistungsfähigkeit von TCP zu erhöhen, sind neue technische Lösungen gefragt. Dabei müssen die bestehenden Technologien erweitert werden, so daß die neue Strukturen mit den bestehenden kompatibel sind bzw. für sie transparent bleiben. So werden z.B. die *Fast-Retransmit* und *Fast-Recovery* Mechanismen, die eine Erweiterung des TCP darstellen, eingesetzt, um den *Slow-Start* Mechanismus zu umgehen. Das *Wireless Application Protocol* (WAP, siehe [WAP]) ist eine weitere, umfassende Möglichkeit, die bereits bestehende Internet-Infrastruktur an mobile digitale Kommunikation anzupassen. WAP bietet einen Rahmen für die Entwicklung der praxistauglichen Anwendungen und gewährleistet Kompatibilität und Zusammenarbeit der von verschiedenen kommerzielen Anbietern betriebenen WAP-Systeme.

1 TCP und drahtlose Kommunikation

Folgende Eigenschaften der drahtlosen digitalen Leitungen tragen zu der geringen TCP-Leistung bei:

- niedrige Bandbreite;
- hohe Fehlerrate;
- Halbduplex-Übertragung;
- häufige Verbindungsabbrüche.

1.1 TCP über langsame Kommunikationkanäle

Als langsam werden in diesem Abschnitt die Datentransportkanäle betrachtet, die nur einen niedrigeren Datendurchsatz ermöglichen, als das Endgerät, das über diese Kanäle mit der Außenwelt kommuniziert, imstande ist zu verarbeiten.

Die Probleme sind vor allem niedrige Datenübertragungsraten und zu lange Antwortzeiten im Fall interaktiver Anwendungen. Eine allgemeine Lösung, um den Datendurchsatz zu erhöhen und die Antwortzeiten zu reduzieren, ist möglichst wenige physische Bits über solchen langsamen Kommunikationkanal zu senden.

Einerseits, man könnte die Menge an Protokoll-Bits, die neben den Daten übertragen werden, reduzieren. Als Beispiel der praktischen Umsetzung dieses Prinzips kann man die Van Jacobsons Header-Kompression nennen, die u.a. in SLIP und PPP ihre Verwendung findet.

Andererseits, ist es wünschenswert, die Menge an Nutzdaten durch Datenkompression zu reduzieren. Dies funktioniert aber nicht immer perfekt. Falls die Daten auf einer oberen Kommunikation-Schicht bereits komprimiert oder verschlüsselt worden sind, enthalten sie keine Redundanz mehr, und eine wiederholte Komprimierung ist nutzlos und führt nur zu Leistungseinbußen. Als Lösungsansatz für die Komprimierung der Nutzdaten nennt [DMKM99a] den „IP Payload Compression Protocol".

Bei der Koppelung der langsamen Kommunikationskanälen an Datennetze mit höherer Übertragungsgeschwindigkeit tritt eine weitere Art von Problemen auf. Der Rechner, der die Daten aus dem Breitband-Netz empfängt und anschließend in den langsamen Datenübertragungkanal sendet, muß aufgrund eines überlaufenden Datenpuffers ankommende Datenpakete verwerfen, was zur Wiederholung der Datenübertragung und einem verringerten Datendurchsatz führt. Diese Besonderheiten können bei der Entwicklung von Applikationen berücksichtigt werden. Durch Änderung der Größe des TCP-Eingangspuffers und eine angepaßte Implementierung der Warteschlange kann die Leistung bedeutend erhöht werden.

1.2 TCP über Leitungen mit hohen Bitfehlerraten

Die Entwicklung des TCP-Protokolls hat auf Festnetzen mit geringen Fehlerraten stattgefunden. Dadurch wurden die Gegebenheiten der Datenkanälen mit hohen Fehlerraten so gut wie nicht berücksichtigt. So wird als einzige Ursache des Verlustes der Datenpakete ein Stau (*congestion*) und damit verbundenes Verwerfen von Paketen (*packets discarding*) angenommen. Um diese Verluste zu behandeln, wurden der Algorithmus zur Stauvermeidung (*congestion avoidance*) und der *Slow-Start* Algorithmus entwickelt.

Um diese Algorithmen zu unterstützen, sind zwei zusätzliche Felder im TCP-Header vorgesehen: *congestion window* (`cwnd`) und *slow start threshold size* (`ssthresh`). Am Anfang der Übertragung wird `cwnd` auf die Segmentgröße und `ssthresh` auf 65535 Bytes gesetzt. Der Sender sendet immer nicht mehr Bytes als Minimum von `cwnd` und TCP-Fenstergröße. Der Empfänger bestätigt den Empfang der Datenpakete mit Empfangsquittung (*acknowlegement, ACK*). Das ACK enthält die Nummer des Pakets, das als nächstes vom Empfänger erwartet wird. Die ACKs können aus Effizienzgrunde verzögert werden. Nach jedem angekommenen ACK wird das `cwnd` erhöht. Diese

Erhöhung hängt vom Wert des `ssthresh` ab. Ist das `cwnd` kleiner als der `ssthresh`, wächst das `cwnd` exponentiell; andererfalls (fast) linear. Kommt ein Paket nicht in der richtigen Reihenfolge an, kann der Empfänger sofort ein sog. (*duplicate ACK*) senden, das wiederum die erwartete Paketnummer enthält. Das Ausbleiben der ACKs oder ggf. der Erhalt *duplicate ACKs* bedeuten einen Paketverlust.

Falls der Sender den Paketverlust entdeckt, setzt er den `ssthresh` auf die Hälfte der gegenwärtigen Fenstergröße, aber mindestens auf 2 Segmentengrößen. Dieses Vorgehen heißt der *Congestion Avoidance* Algorithmus. Wurde der Paketverlust durch ein Timeout angezeigt, so wird außerdem das `cwnd` auf die Segmentgröße reduziert, und dadurch der *Slow-Start* Algorithmus aktiviert. Falls die Paketenverluste nicht durch Stau, sondern durch hohe Bitfehlerrate zustandekommen, ist es sehr wahrscheinlich, daß weitere Pakete verlorengehen. Durch Wirkung der obengenannten Algorithmen bleibt dann die TCP-Fenstergröße und schließlich die Übertragungsrate extrem klein.

Es gibt leider noch keine etablierten und von der Allgemeinheit anerkannten Ansätze, die Paketverluste, die durch hohe Bitfehlerraten entstehen, zu behandeln. Als Problemlösungen kommen für die Autoren von [DMKM+99b] außer einigen Studien, die sich noch in Experimentellphase befinden, lediglich die Verwendung der *Fast-Retransmit* und *Fast-Recovery* Algorithmen in Frage.

Für den *Fast-Retransmit* Algorithmus werden drei gleiche *duplicate ACKs* als Anzeige für Paketverlust betrachtet. In solchem Fall wird das Segment, das vom Empfänger erwartet wird, sofort gesendet, und nicht wie im „Normalfall“ auf Retransmit-Timeout gewartet. Das spart Zeit und erhöht den Datendurchsatz.

Für die Verwendung des *Fast-Recovery* Algorithmus spricht die Tatsache, daß die *duplicate ACKs* nur dann zustandekommen, wenn bereits einige Pakete an den Empfänger angekommen sind. So ist es nicht wünschenswert, die Fenstergröße auf Segmentgröße, wie beim *Slow-Start*, zu verkleinern. In diesem Fall wird das fehlende Paket gesendet, der `ssthresh` auf die Hälfte des `cwnd` reduziert, und das `cwnd` nicht auf die Segmentgröße sondern auf `ssthresh` plus dreifache Segmentgröße gesetzt. Nach erfolgter Bestätigung wird das `cwnd` auf Größe des `ssthresh` gesetzt und weiter (jetzt im *Congestion Avoidance* Modus) fortgefahren. Kommen dagegen weitere *duplicate ACKs* an, wird das `cwnd` um eine Segmentgröße erhöht, und der Sender wiederholt das verlorengegangene Paket, falls die `cwnd` Größe dies erlaubt.

Die *Fast-Retransmit* und *Fast-Recovery* Algorithmen sind zwar für die Stauvermeidung entwickelt worden, erreichen aber auch im Fall der Kanalfehler eine Durchsatzsteigerung. Leider funktionieren diese Algorithmen bei kleiner TCP-Fenstergröße nicht (es kommen einfach keine 3 *duplicate ACKs* zusammen), was unweigerlich zu *Slow-Start* führt.

1.3 TCP über asymmetrische Leitungen

Die Probleme, die bei TCP-Übertragung auf asymmetrischen (z.B. ADSL, Satellit-Download) Leitungen, sowie auf Leitungen, die im Halb-Duplex Modus betrieben werden (die meisten Radio-Verbindungen), sind gleich. Bei solchen Übertragungen gehen die ACKs oft verloren, oder sie kommen zu spät an. Für den Sender sieht dies wie ein Verlust der gesendeten Pakete aus. Die eigentlich erfolgreich empfangene Pakete

werden erneut übertragen, außerdem werden die im Abschnitt 1.2 besprochene *Slow-Start* und *Congestion Avoidance* Algorithmen aktiviert, was zusätzlich und unnötig Zeit verschwendet.

Die Lösungen existieren in zwei Richtungen. Erstens, ist es möglich, die Verluste von ACKs auf dem Rückweg zu reduzieren. Das kann erreicht werden durch:

- *Header compression* (siehe Abschnitt 1.1).
- *ACK filtering.* Hier wird die kumulative Natur der ACKs ausgenutzt. Nach der Ankunft eines ACK an einem Router, kann dieser überprüfen, ob in seiner Warteschlange die ACKs mit kleineren Paketnummern für dieselbe Verbindung enthalten sind, und ggf. werden diese aus der Warteschlange entfernt.
- *ACK-first scheduling.* Router kann verschiedene Warteschlangen für Datenpakete und Pakete mit ACKs unterhalten. Die ACKs, falls vorhanden, haben dann eine höhere Priorität.
- *ACK congestion control.* Die Idee dieses Ansatzes ist, den Stau der ACKs auf dem Rückweg zu vermeiden. Der Router an der Verbindung, die den Flaschenhals darstellt, beobachtet die Länge seiner Warteschlange, und meldet ggf. dem Empfänger den ACK-Stau mittels eines gesetzten ECN-Bits (*explicit congestion notification*) im Header eines Pakets.

Die zweite Lösungsmöglichkeit ist, verlorene oder verspätete ACKs zu berücksichtigen und zu versuchen, unerwünschte Effekte zu vermeiden:

- *TCP sender adaptation.* Bei diesem Ende-zu-Ende Ansatz bestätigt der Empfänger nicht jedes einzelnen angekommenen Paket, sondern gleich mehrere. Dadurch wird die Anzahl der ACKs reduziert. Der Sender ändert dann die Größe seines TCP-Fensters entsprechend der Anzahl implizit bestätigter Pakete.
- *ACK reconstruction.* Hier werden die gefilterte oder verlorengegangene ACKs durch einen Router wiederhergestellt. Dadurch wird das *Congestion Window* beim Sender entsprechend der Anzahl empfangener Pakete erhöht. Dieser Ansatz ist besonders für ISPs geeignet und fordert vom Sender die Unterstützung für *ACK filtering* und für *TCP sender adaptation* nicht.

2 Wireless application protocol (WAP)

1997 haben sich mehrere Mobilkommunikations-Anbieter und Hersteller mobiler Geräte zusammengeschlossen und das *Wireless Application Protocol Forum (WAP forum)* gegründet (siehe [WAP]). Das Hauptziel dieses Gremiums ist, verschiedene Internet-Inhalte, z.B. WWW-Seiten, und andere Informationsdienste auf mobile Telefone und andere digitale mobile Geräte (PDAs, Notebooks usw.) zu bringen. Außerdem sollen die entwickelte Protokolle globale Kommunikation zwischen verschieden Mobilnetz-Typen (GSM, CDPD, UMTS usw.) ermöglichen. Entstehende Lösungen sollen folgenden Anforderungen entsprechen:

- **Interoperabilität.** Mobile Geräte und Software von verschiedenen Hersteller sollen in der Lage sein, mit Mobilnetzwerken anderer Anbieter zu kommunizieren.

- **Skalierbarkeit.** Protokolle und Dienste sollen skalierbar über die Anzahl von Benutzer und deren Ansprüche sein.

- **Effizienz.** Sicherstellung der passenden Qualität der Kommunikation entsprechend den Eigenschaften des Netzwerks.

- **Zuverlässigkeit.** Sicherstellung der konsistenten und vorhersehbaren Basis für die Entwicklung der Netzdienste.

- **Sicherheit.** Schutz der Benutzerdaten und Geräte vor Sicherheitproblemen.

2.1 Architektur

In Abbildung 1 wird die WAP Architektur skizziert und mit der Architektur eines typischen Internet-Dienstes, dem WWW, verglichen.

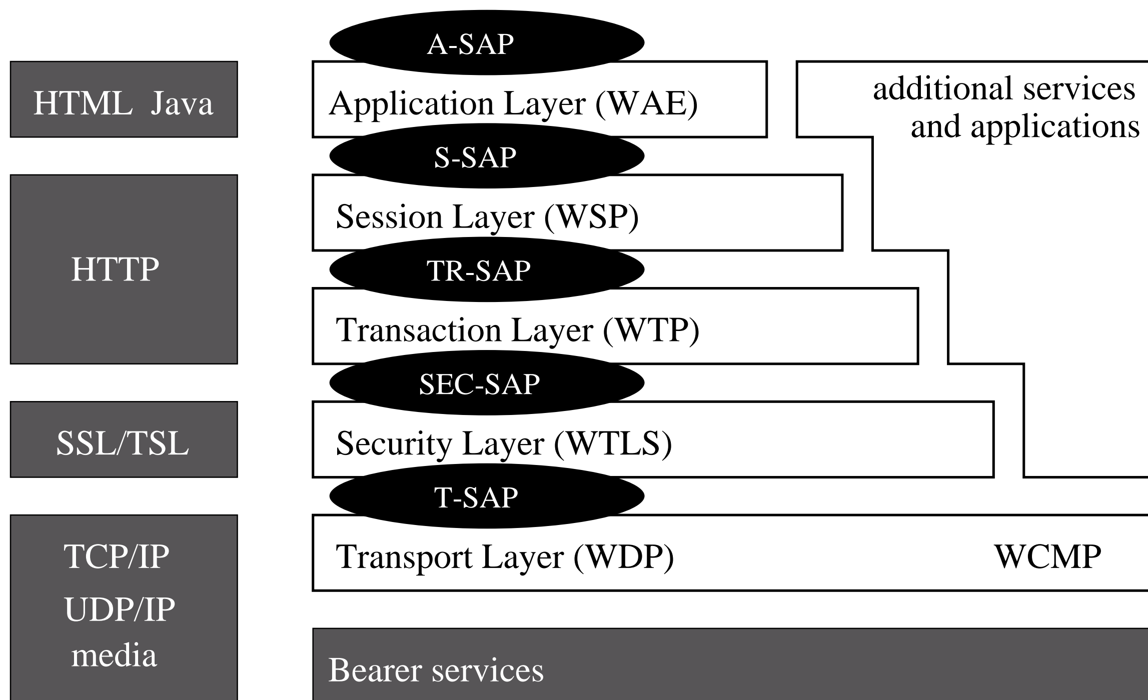


Abbildung 1: Komponente und Schnittstellen der WAP-Architektur

Die Grundlage für die Datenübertragung sind die Übertragungdienste (*Bearer Services*). WAP spezifiziert diese Dienste nicht, sondern benutzt bereits existierende Datenübertragung-Dienste, z.B. SMS (*Short Message Service*) oder GPRS (*General Packet Radio Service*) von GSM oder CSD (*Circuit Switched Data*). Weiterhin werden auch die neu entstehende Dienste in diese Schicht integriert. Es wird kein Dienstzugangspunkt zu dieser Schicht definiert, da die Implementierung vom Trägertyp abhängig ist.

Die Transport-Schicht (*Transport Layer*) mit ihrem *Wireless Datagram Protocol (WDP)* und zusätzlichem *Wireless Control Message Protocol (WCMP)* stellt den höheren Protokollschichten einen konsistenten und von Trägertyp unabhängigen Datagram-orientierten Dienst zur Verfügung. Als Dienstzugangspunkt zu diesem Schicht dient *Transport Layer Service Access Point (T-SAP)*.

Die Sicherheitschicht (*Security Layer*) mit dem *Wireless Transport Layer Security protocol (WTLS)* ermöglicht kryptographischen Schutz der übertragenden Daten, Autentifizierung und den Schutz vor *Denial-Of-Service* Attacken. Das WTLS Protokoll basiert auf TLS (*Transport Level Security*), früher SSL (*Secure Sockets Layer*), dem Protokoll, das in WWW verwendet wird. WTLS wurde für die Nutzung über drahtlose und schmalbandige Kanäle optimiert. Der Dienstzugangspunkt zu dieser Schicht heißt *SECurity SAP (SEC-SAP)*.

Die Transaktionschicht (*Transaction Layer*) mit dem *Wireless Transaction Protocol (WTP)* stellt einen Transaktionsdienst zur Verfügung. Dieser Transaktionsdienst ist auf einen geringen Kommunikationaufwand ausgelegt und ermöglicht gesicherte (*relaible requests*) und ungesicherte (*unreliable requests*) Anfragen, sowie asynchrone Transaktionen. Der Dienstzugangspunkt heißt *TRansaction SAP (TR-SAP)*.

Die Sitzungschicht (*Session Layer*) mit ihrem *Wireless Session Protocol (WSP)*, stellt zu diesem Zeitpunkt zwei Dienste zur Verfügung. Der erste ist verbindungorientiert (*connection-oriented service*) und ist eng mit darunterliegender *Transaction Layer* verbunden. Der andere ist nicht auf Verbindungen ausgerichtet (*connectionless service*) und interagiert direkt mit der Transport-Schicht. Außerdem ist ein spezieller Dienst (*WSP/B*) für Nutzung der WWW definiert worden. Der Dienstzugangspunkt heißt *Session-SAP (S-SAP)*.

Die oberste Schicht des Protokollstacks ist die *Application Layer* mit dem *Wireless Application Environment (WAE)*. Diese Schicht definiert einen Rahmen für die Einbindung verschiedener WWW- und Mobilfunk-Applikationen. Der Akzent liegt hier auf speziellen Skriptsprachen und *Markup Languages*, sowie Schnittstellen für Mobilfunk-Anwendungen.

Es ist für die Anwendungen nicht erforderlich, alle Bestandteile der WAP-Architektur zu verwenden. Eine Anwendung, die z.B. keine Dienste der Sicherheitschicht benötigt, kann direkt auf die Dienste der Transport-Schicht zugreifen.

2.2 Wireless datagram protocol (WDP)

Mit seinem Dienstzugangspunkt T-SAP offeriert das WDP einen konsistenten Dienst für den Datagram-Transport. Für die Benutzer dieses Dienstes bleibt der Austausch von Datagrammen transparent, d.h. die Besonderheiten des darunterliegenden Datenübertragungdienstes (*beared service*) bleiben verborgen. Je ähnlicher dieser Datenübertragungdienst zu IP ist, desto weniger Anpassungen sind nötig. Falls der Datenübertragungdienst bereits IP zur Verfügung stellt, wird UDP für WDP benutzt.

Das Protokolldiagramm wird in Abbildung 2 dargestellt. Die Parameter `DA` (*destination address*), `DP` (*destination port*), `SA` (*source address*), `SP` (*source port*) und `UD` (*user data*) sind verpflichtend. `DA` und `SA` sind eindeutige Identifikatoren für Sender und

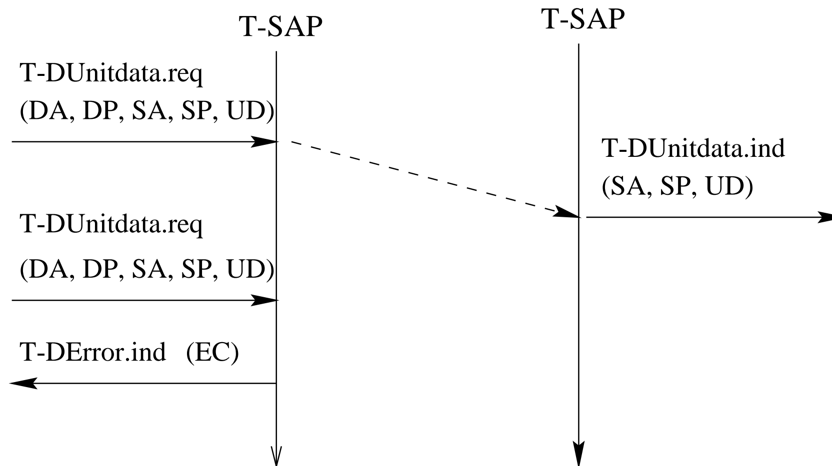


Abbildung 2: WDP-Dienstprimitiven

Empfänger der Daten. Das können z.B. die Telefonnummern oder IP-Adressen sein. Für `T-DUnitdata.ind` sind `DA` und `DP` optional.

Falls der Auftrag zur Datenübertragung seitens WDP nicht ausgeführt werden kann, wird ein Fehler mit dem Primitiv `T-DError.ind` gemeldet. `EC` (*error code*) enthält den Fehler-Code. Dieses Primitiv soll ausschließlich zum Anzeigen der lokalen Fehler benutzt werden. Übertragungfehler werden mit Hilfe des Zusatzprotokolls WCMP (*Wireless Control Message Protocol*) behandelt.

2.3 Wireless transport layer security (WTLS)

Die Sicherheits-Schicht (*Security Layer*) kann über die Transport-Schicht (*Transport Layer*) eingesetzt werden. WTLS ermöglicht Sicherheit der Verbindungen, Geheimhaltung der übertragenden Daten und die Authentifizierung. Das entsprechende Protokoll, *Wireless Transport Layer Security (WTLS)*, wurde von TLS abgeleitet und an die Gegebenheiten der mobilen Kommunikation (niedrige Bandbreite, niedrige Rechenleistung und Speichergröße der mobilen Geräte usw.) angepaßt.

Der Verbindungaufbau beginnt mit dem WTLS-Handshake (Abbildung 3). Der Teilnehmer, der die Verbindung aufbaut wird als Sender (*originator*) und der andere als Empfänger (*peer*) bezeichnet. Die Parameter fürs `SEC-Create.req` Primitiv sind: `SA` (*source address*), `SP` (*source port*), `DA` (*destination address*), `DP` (*destination port*), `KES` (*key exchange suite*, z.B. RSA, Diffie-Hellman), `CS` (*cipher suite*, z.B. DES, IDEA) und `CM` (*compression method*, zur Zeit nicht spezifiziert); für `SEC-Create.res`: `SNM` (*sequence number mode*), `KR` (*key refresh cycle*, d.h. wie oft muß der Schlüssel während der Sitzung geändert werden), `SID` (*session identifier*, eindeutig für jeden Kommunikationsteilnehmer), und vom Peer Ausgewählten `KES'`, `CS'` und `CM'`. Mit dem `SEC-Exchange.req` teilt der Peer dem Originator mit, daß er Public-Key Authentifizierung wünscht. Jeder Beteiligte kann in einer beliebigen Phase den Prozeß abbrechen, etwa wenn die angekommenen Parameter nicht akzeptiert wurden.

Der eigentliche Austausch von Datagrammen wird mit einfachen `SEC-Unidata.*` Primitiven abgewickelt (Abbildung 4). Diese Primitiven haben die gleiche Funktionen, wie `T-Unidata.*` Primitiven von WDP. Die obere Schichten können deshalb die SEC-SAP Schnittstelle statt T-SAP benutzen.

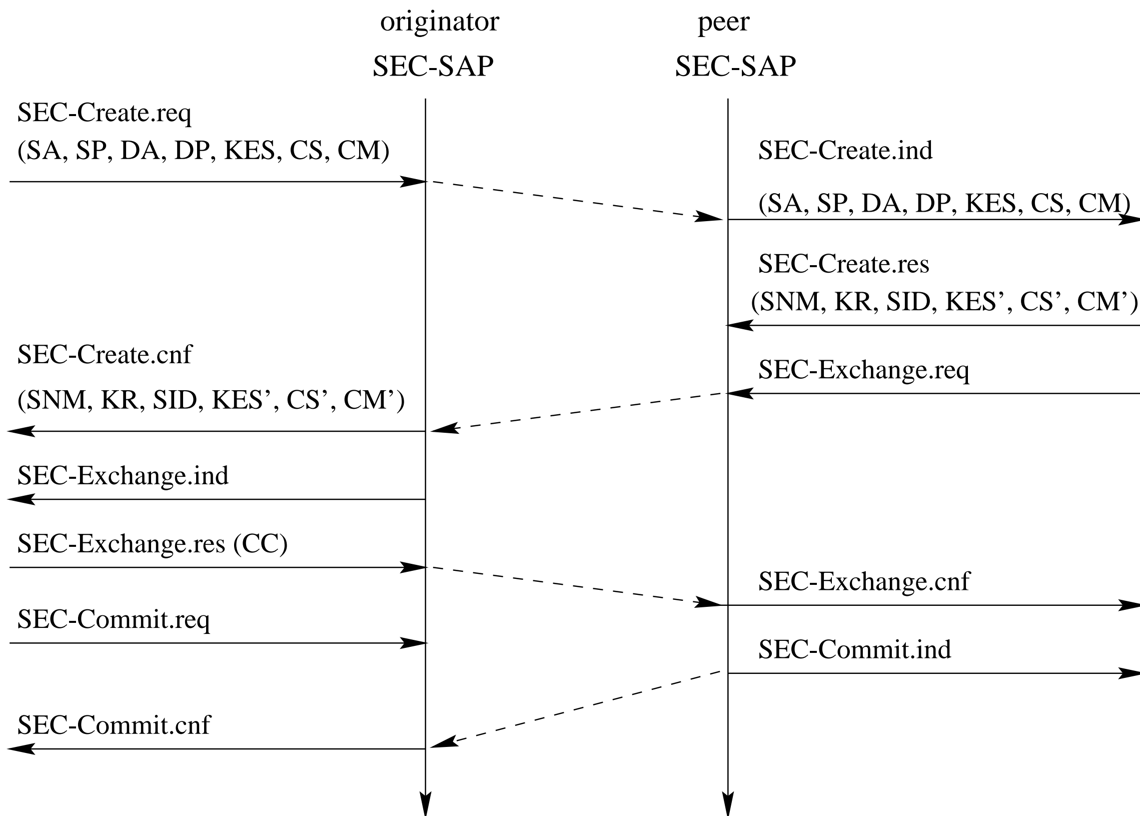


Abbildung 3: Aufbau einer WTLS-Sitzung

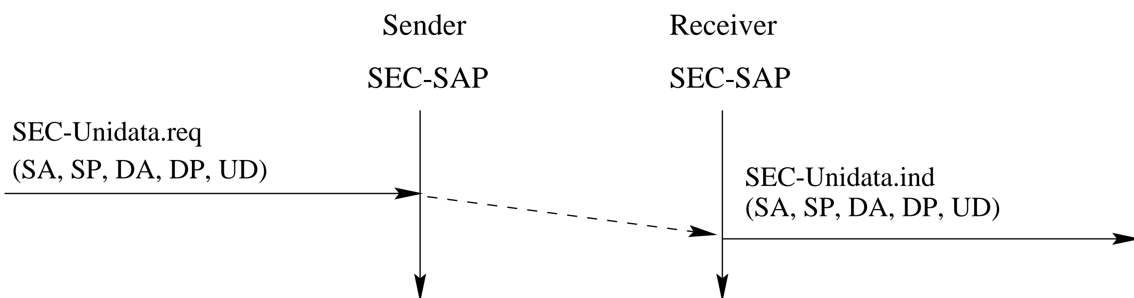


Abbildung 4: Übertragung eines WTLS-Datagrams

2.4 Wireless transaction protocol (WTP)

Das *Wireless Transaktion protocol (WTP)* benutzt das darunterliegenden WDP oder (falls die Sicherheitsmaßnahmen nötig sind) WTLS. WTP ist nicht auf Byte-Streams, sondern auf einzelne Nachrichten ausgelegt. WTP wird in 3 Klassen aufgeteilt. Die Klasse 0 bietet eine ungesicherte Übertragung ohne Quittungen. Die Klassen 1 und 2 gewährleisten die Zuverlässigkeit der Übertragung. WTP sieht keine expliziten Auf- und Abbauphasen vor. Dadurch bleibt unnötiger Aufwand erspart. Die Zuverlässigkeit wird durch folgende Maßnamen erreicht:

- Entfernung der Duplikate (*duplicate removal*)
- Übertragungwiederholung (*retransmission*)
- Empfangsbestätigung (*acknowledgements*)

- Verwendung eindeutiger Transaktion-Identifikatoren (*transaction identifiers*)

WTP stellt für höhere Schichten u.A. folgende Möglichkeiten zur Verfügung:

- asynchrone Transaktionen (*asynchronous transaction*);
- Abbruch der Transaktion (*abort of transaction*);
- Zusammenhängen der Nachrichten (*concatenation of messages*);
- Bestätigung der Ankunft oder Benachrichtigung über Verlust einer Nachricht während der gesicherten Übertragung(*report success or failure of reliable messages*).

Eine spezielle Eigenschaft von WTP ist die Benutzerbestätigung (*user acknowledgement*) bzw. automatische Bestätigung (*automatic acknowledgement*) durch WTP-Entität. Falls die Benutzerbestätigung gefordert wird, muß der Benutzer jede empfangene Nachricht bestätigen. Dies gewährleistet, daß die Nachricht tatsächlich den WTP-Benutzer erreicht hat und von ihm bearbeitet wird.

2.4.1 WTP Klasse 0

Die WTP Klasse 0 bietet einen Dienst für ungesicherte Transaktionen ohne Quittungen. Diese Transaktionen sind zustandslos und können nicht abgebrochen werden Das Transaktiondiagramm wird in Abbildung 5 dargestellt.

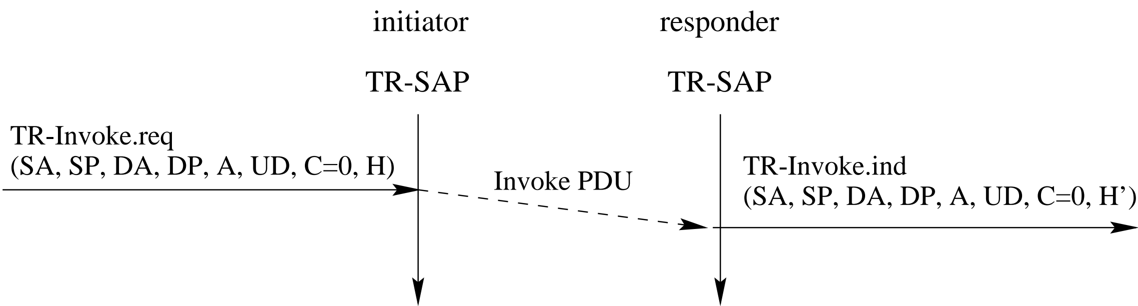


Abbildung 5: Grundtransaktion, WTP Klasse 0

Die Parameter `SA`, `SP`, `DA`, `DP` und `UD` wurden bereits im Abschnitt 2.2 erläutert. Mit dem Flag `A` (*acknowledgement*) bestimmt der Benutzer des Dienstes, ob eine Bestätigung vom gegenüberliegenden WTP-Benutzer erforderlich ist (Für Klasse 0 ist dies nicht der Fall). Der Typ der WTP-Klasse wird in `C` (*class type*) angegeben (hier 0). Die Transaktionreferenz `H` bzw. `H'` (*transaction Handle*) stellen eine Zusammenfassung der Tupel (SA, SP, DA, DP) dar und identifizieren eindeutig die Transaktion auf entsprechender Seite (Stichwort „Socket"). Die Klasse 0 sieht weder eine Empfangsbestätigung noch eine Übertragungwiederholung vor. Das Verfahren ähnelt sehr jenem von WDP, und soll benutzt werden, falls lediglich eine Datagram-Übertragung stattfinden soll.

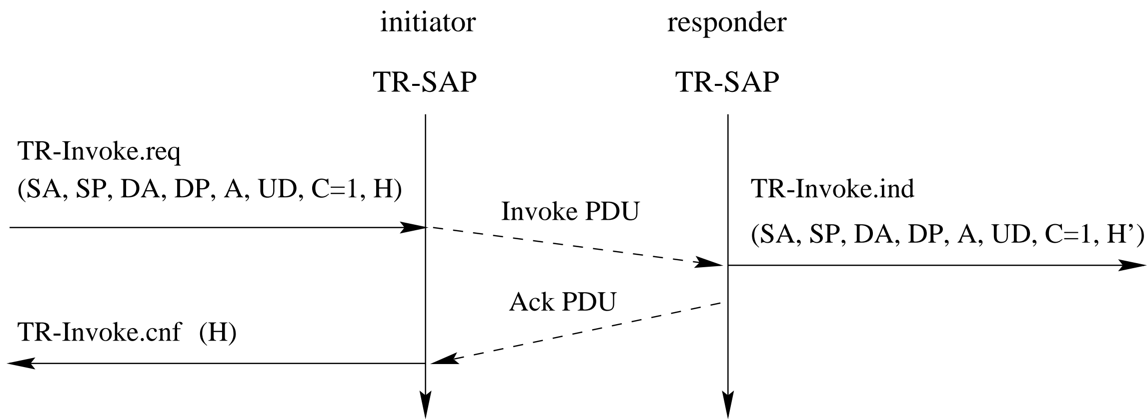


Abbildung 6: Grundtransaktion, WTP Klasse 1, ohne Benutzerbestätigung

2.4.2 WTP Klasse 1

Die WTP Klasse 1 bietet einen Dienst zur Durchführung gesicherter Transaktionen (Abbildung 6). In diesem Fall ist `C` gleich 1, und es wird wieder keine Benutzerbestätigung angefordert. Die Empfänger-Seite (*responder*) sendet automatisch eine Bestätigung an Sender (*initiator*) zurück. Die Spezifikation erlaubt es dem Benutzer auf der Empfänger-Seite, auch eine Benutzerbestätigung zu senden, verpflichtet ihn aber nicht. Der Empfänger behält den Transaktionzustand, um im Fall der wiederholten Ankunft der gleichen Nachricht (was Verlust der Bestätigung bedeutet) in der Lage zu sein, die Bestätigung zu wiederholen.

Das Diagramm für den Fall einer angeforderten Benutzerbestätigung wird in Abbildung 7 dargestellt. Die Empfänger-Seite sendet keine automatische Bestätigung, sondern wartet auf das `TR-Invoke.res` Primitiv vom Benutzer. Dieses Primitiv muß ein entsprechendes `H'` haben, um die Transaktion richtig zu identifizieren.

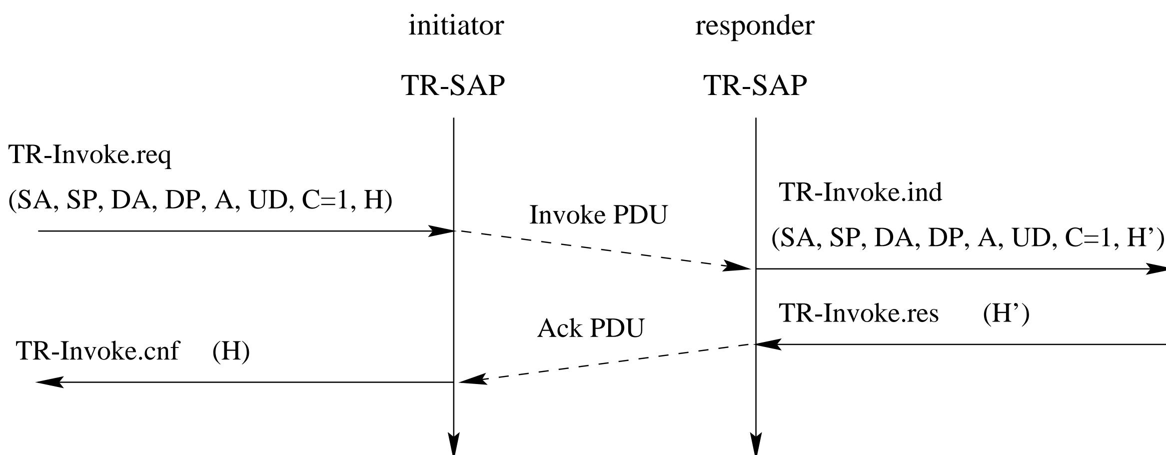


Abbildung 7: Grundtransaktion, WTP Klasse 1, mit Benutzerbestätigung

2.4.3 WTP Klasse 2

Die WTP Klasse 2 führt klassische gesicherte Anfrage/Antwort Transaktionen durch, wie sie von zahlreichen Klient/Server Szenerien bekannt sind. Es sind mehrere Varianten im Transaktionablauf abhängig von den gestellten Anforderungen möglich. Hier werden drei Beispiele vorgestellt.

Die Abbildung 8 präsentiert die Grundtransaktion der Klasse 2 ohne Benutzerbestätigung. Eine weitere, noch zuverlässigere Variante der Transaktion wird in Abbildung 9 dargestellt. Hier bestätigt der Benutzer auf Empfänger-Seite die Ankunft der Nachricht explizit. In beiden Fällen wird der Empfänger über die Transaktionergebnisse benachrichtigt.

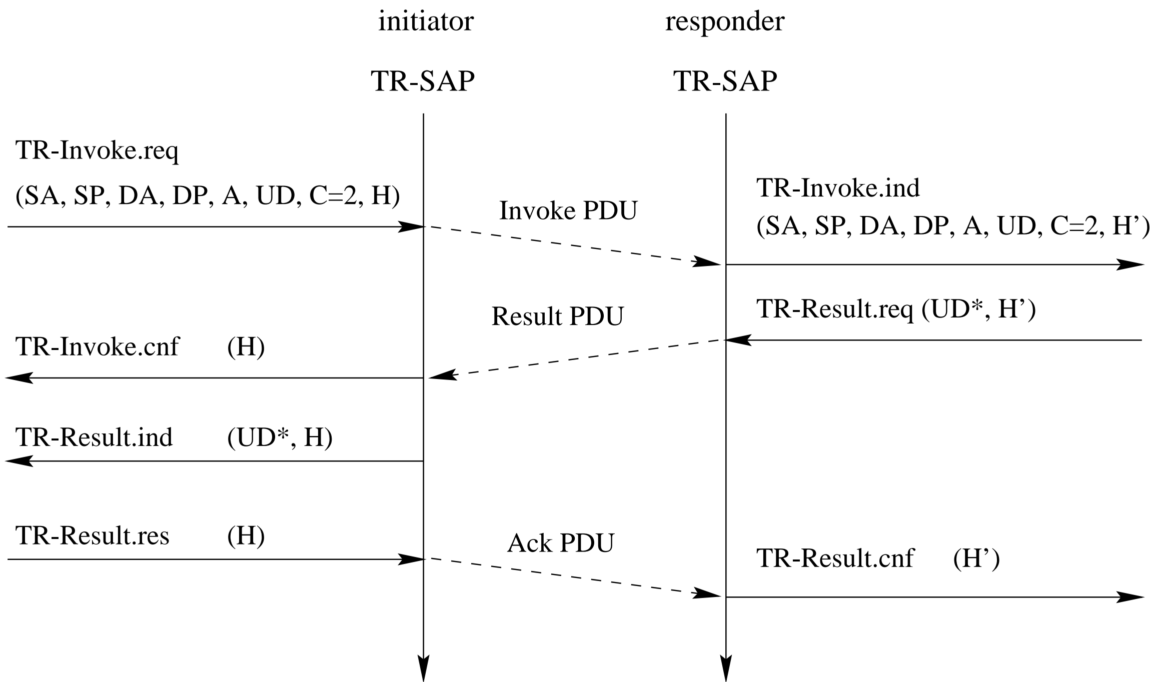


Abbildung 8: Grundtransaktion, WTP Klasse 2, ohne Benutzerbestätigung

Falls die Bearbeitung der Anfrage vom Benutzer auf der Empfänger-Seite zu lange dauert und keine Benutzerbestätigung angefordert wurde, kann es dazu führen, daß innerhalb einer definierter Zeitspanne keine implizite Bestätigung des Empfangs eintrifft. Dann nimmt die Sender-Seite an, daß die Nachricht verloren ging, und sendet sie dann erneut an den Empfänger. Um diesen unerwünschten Effekt vorzubeugen, kann der Empfänger den Sender in den „*hold on*“ Zustand versetzen. Das wird dadurch erreicht, daß nach einem bestimmten Timeout die Empfänger-Seite automatisch den Empfang bestätigt (Abbildung 10).

WTP bietet viele weitere Möglichkeiten an, Transaktionen durchzuführen, die hier nicht behandelt werden. Dazu zählt z.B. das Aneinanderhängen von Nachrichten, Verwaltung der asynchronen Transaktionen mit bis zu 215 ausgesetzten (*outstanding*) Transaktionen (angefordert aber bis jetzt noch nicht abgeschlossen), und das Fragmentieren und Reassemblieren von Nachrichten.

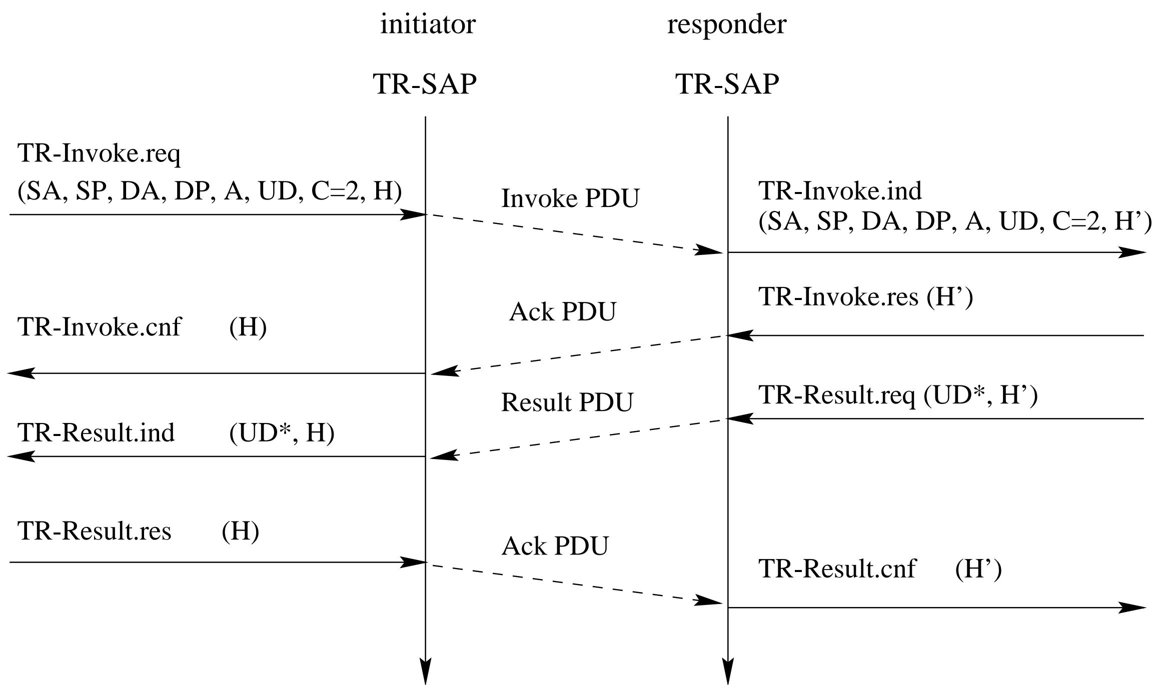


Abbildung 9: Grundtransaktion, WTP Klasse 2, mit Benutzerbestätigung

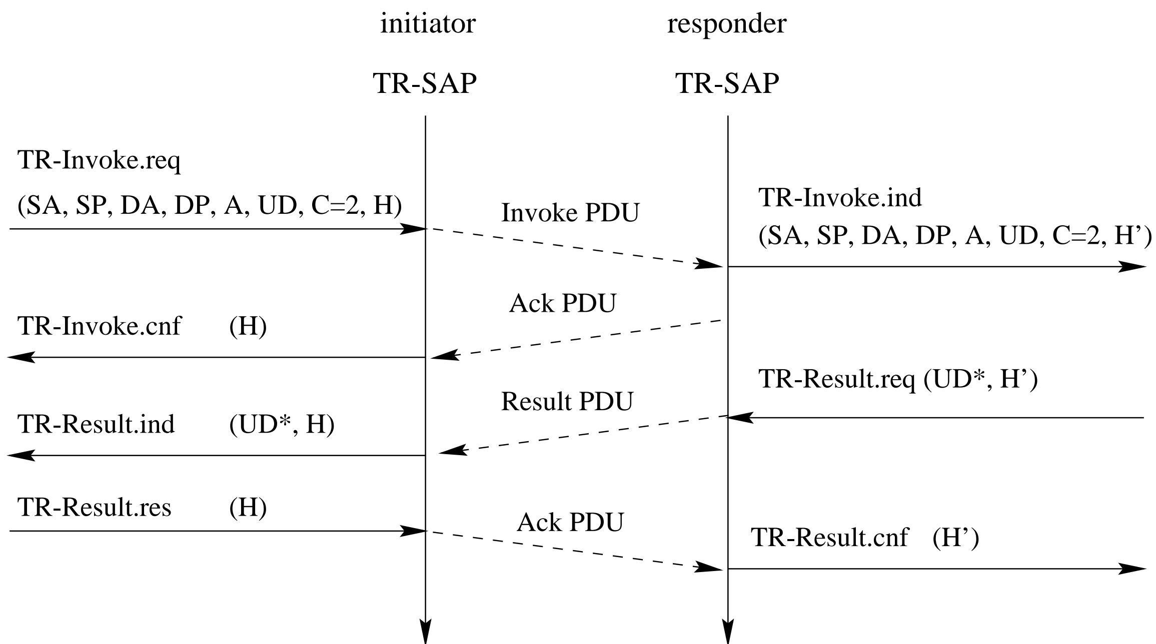


Abbildung 10: WTP Klasse 2 mit *"hold on"*, ohne Benutzerbestätigung

2.5 Wireless session protocol (WSP)

Die Grundidee der Einführung des WSP ist die Notwendigkeit zustandsbehafteter Sitzungen. Das Vorhandensein solcher Sitzungen ist sehr hilfreich, um z.B. die Übertragung einer Seite genau an der Stelle fortzusetzen, an der sie unterbrochen wurde. WSP bietet folgende Möglichkeiten:

- Sitzungverwaltung (*session management*): Die Sitzung kann vom Klient zum Server geöffnet (*established*)werden und wieder ordentlich geschlossen(*released*)

werden. Eine Sitzung kann vorübergehend ausgesetzt und wiederaufgenommen werden (*suspend and resume*). Die Lebensdauer einer Sitzung ist von Lebensdauer der Transport-Verbindung oder der Dauer der Operationen auf dem Träger unabhängig und kann sehr groß sein.

- Anpassen der gemeinsam verwendbaren Funktionalität und der Verbindungeigenschaften (*capability negotiation*): Klienten und Server einigen sich während der Eröffnung der Sitzung über die gemeinsam benutzte Protokoll-Funktionalität.
- Inhaltskodierung (*content encoding*): Der WSP definiert effiziente binäre Kodierung für die zu übertragenden Inhalte.

Für die Web-Anwendungen wird eine Spezialisierung des allgemeines WSP-Protokolls, WSP/B (*wireless session protocol/browsing*) definiert. WSP/B bietet zusätzlich an:

- HTTP/1.1 Funktionalität: WSP/B ist im Grunde eine binäre Form von HTTP/1.1 und unterstützt seine Funktionalität. Die Header von HTTP/1.1 werden für die Definition des Inhalttyps benutzt, wobei für verbreitete Header binäre Codes definiert werden.
- Austausch von Sitzungs-Headern: Klient und Server können die Header, die während der Sitzung unverändert bleiben, einmal am Anfang der Sitzung austauschen, um sie später zu referenzieren. WSP/B interpretiert diese Header nicht, sondern gibt sie an die Benutzer weiter.
- Push und Pull Übertragungstrategien: zusätzlich zu dem traditionell im Web benutzen Pull-Mechanismus bietet WSP/B drei zusätzliche Push-Mechanismen: bestätigter und nicht bestätigter Push innerhalb eines existierenden Sitzungkontext, und nicht bestätigter Push außerhalb eines existierenden Sitzungkontextes.
- Asynchrone Anfragen: WSP/B unterstützt optional Klienten, die an Server mehrere Anfragen gleichzeitig stellen können. Dies erhöht die Effizienz der Anfragen und Antworten, weil sie zu einigen wenigen Nachrichten zusammengefügt werden können. Die Latenz wird ebenfalls verringert, weil die Ergebnisse sofort nach ihrer Verfügbarkeit abgesandt werden können.

2.6 Wireless application environment (WAE)

Das Ziel des WAE ist es, eine standardisierte Umgebung für WAP-Anwendungen zu bieten. Diese Umgebung basiert auf bereits bestehenden Technologien und berücksichtigt die Eigenschaften der drahtloser Kommunikation.

Die WAE-Hauptelemente sind:

- *Wireless Markup Language (WML).* Die WML ist vom HTML abgeleitet. Ein WML-Dokument besteht aus Stapeln (*decks*), die ihrerseits aus Karten (*cards*) bestehen. Eine Karte soll leicht z.B. auf einem Handy-Bildschirm darstellbar sein. Zusätzlich zu den aus HTML bekannten Eigenschaften, sieht die WML die Möglichkeit vor, den Zustand der Umgebung beim Stapel-Wechsel ohne Interaktion mit dem Server zu speichern.

- *WMLScript*. Diese Skriptsprache basiert auf JavaScript und wurde an die drahtlose Kommunikation angepaßt. WMLScript ist ereignisorientiert und bietet Zugriff auf WML-Variablen.

- *Wireless Telephony Application (WTA)* WTA erweitert WAE um für Telefonie spezifische Funktionalität. Die WTA-Anwendungen benutzen das *Wireless Telephony Application Interface (WTAI)* und bieten dem Benutzer die Möglichkeit, sein mobiles Gerät (das nicht unbedingt ein Handy sein muß) als Telefon zu benutzen.

3 Zusammenfassung

Es existieren mehrere Ansätze, um die Leistungsfähigkeit von TCP auf drahtlosen Kanälen zu erhöhen. Der Datendurchsatz kann durch Kompression der Nutz- und Kontrolldaten erhöht werden. Die durch die TCP-Eigenschaften hervorgerufenen Leistungseinbußen durch Paketverlusten können durch Verwendung von *Fast-Retransmit* und *Fast-Recovery* Algorithmen vermieden werden. Die Lösungansätze zur Leistungssteigerung im Fall einer asymmetrischen oder Halbduplex-Verbindung bauen auf der Verringerung der ACK-Verluste bzw. auf der Wiederherstellung verlorener ACKs auf.

Die vom WAP-Forum entwickelte WAP-Architektur ermöglicht es, Mobilfunk-gerechte Anwendungen und Systeme zu entwickeln. Die Transport-Schicht des WAP bietet einen UDP-ähnlichen Transportdienst, der die Besonderheiten des benutzten Trägers vor dem Benutzer verbirgt. Die Sicherheit-Schicht schützt die Daten vor unbefugtem Zugriff und das System vor DoS-Angriffen. Die Transaktion-Schicht bietet 3 Protokollklassen für verschiedene Arten der Übertragung. Die Klasse 0 bietet ungesicherte Transaktionen und kann immer dort verwendet werden, wo im „klassischen“ Internet das UDP zum Einsatz kommt; z.B. für die Realzeit-Übertragung der Sprache. Die Klassen 1 und 2 bieten sichere Transaktionen und werden verwendet, falls die Integrität der übertragenden Daten geprüft werden muß. Die Klasse 1 ermöglicht die Datenübertragung nur in einer Richtung; Klasse 2 steht für das klassische Anfrage-Antwort Szenario und ermöglicht die Übertragung in beiden Richtungen. Diese beide Klassen können z.B. bei der Übertragung einer WWW-Seite benutzt werden. Die Sitzung-Schicht hat kein TCP/IP-Gegenstück und ermöglicht es, die Sitzungen über einen längeren Zeitintervall aufrechtzuerhalten, ohne eine darunterliegende Transportverbindung.

Die WAP-Architektur berücksichtigt die bereits erwähnte Probleme mit TCP auf drahtlosen Kanälen. Außerdem wird auch die geringe Leistung und beschränkte Darstellungmöglichkeiten der mobilen Geräte berücksichtigt. WAP legt einen großen Wert auf Kompatibilität und ermöglicht die Integration der Lösungen von verschiedenen kommerzielen Anbietern.

Literatur

[BaPa99] Hari Balakrishnan und Venkata N. Padmanabhan. TCP Performance Implications of Network Asymmetry. *Internet Draft „draft-ietf-pilc-asym-00.txt“*, September 1999.

[Brad89] R. Braden (Hrsg.). RFC 1122: Requirements for Internet Hosts – Communication Layers. Request For Comments, Network Working Group, Oktober 1989.

[DMKM99a] S. Dawkins, G. Montenegro, M. Kojo und V. Marget. Performance Implications of Link-Layer Characteristics: Slow links. *Internet Draft „draft-ietf-pilc-slow-01.txt“*, September 1999.

[DMKM+99b] S. Dawkins, G. Montenegro, M. Kojo, V. Marget und N. Vaidya. Performance Implications of Link-Layer Characteristics: Links with Errors. *Internet Draft „draft-ietf-pilc-error-01.txt“*, September 1999.

[KFTM99] Phil Karn, Aaron Falk, Joe Touch und Marie-Jose Montpetit. Advice for Internet Subnetwork Designers. *Internet Draft „draft-ietf-pilc-link-design-00.txt“*, Juni 1999.

[Schi99] Jochen Schiller. *Mobile communications*, Kapitel 11, S. 256–283. Addison-Wesley. 1999.

[Stev97] W. Stevens (Hrsg.). RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. Request For Comments, Network Working Group, Januar 1997.

[WAP] www.wapforum.org.

Abbildungsverzeichnis

1 Komponente und Schnittstellen der WAP-Architektur 49

2 WDP-Dienstprimitiven 51

3 Aufbau einer WTLS-Sitzung 52

4 Übertragung eines WTLS-Datagrams 52

5 Grundtransaktion, WTP Klasse 0 53

6 Grundtransaktion, WTP Klasse 1, ohne Benutzerbestätigung 54

7 Grundtransaktion, WTP Klasse 1, mit Benutzerbestätigung 54

8 Grundtransaktion, WTP Klasse 2, ohne Benutzerbestätigung 55

9 Grundtransaktion, WTP Klasse 2, mit Benutzerbestätigung 56

10 WTP Klasse 2 mit *“hold on”*, ohne Benutzerbestätigung 56