Entwurf

Gesellschaft für
Silizium-Anwendungen
und CAD/CAT
Niedersachsen

SICAN

Entwurf eines Media-Stream Management-Systems für verteilte Media-Server

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Systeme, die auf Abruf kontinuierliche Datenströme verschiedener Medienklassen bereitstellen (On-Demand-Systeme), sind in vielfältiger Weise bereits im Einsatz. Stark vertreten sind zur Zeit auch noch analoge Systeme. Die globale Vernetzung von Rechnern und die Möglichkeiten der Digitaltechnik, mit all ihren Vorzügen, erweist sich als neuer Motor für diese Art der Anwendungen. Wo analoge Systeme schnell an ihre Grenzen stoßen und nur mit hohem Aufwand erweitert werden können, zeigt sich, daß mit Hilfe der Digitaltechnik neue Freiräume geschaffen werden können; Freiräume die den angestrebten Weg erleichtern, unterschiedliche Dienste in einem Endgerät zu integrieren, so wie sich die Integration verschiedenartiger Netze in einem allgemeineren Netz vollziehen wird.

Die steigende Popularität digitaler On-Demand-Systeme führt jedoch auch hier schnell an Grenzen des zur Zeit technisch Machbaren. Damit erzeugt die Problemstellung, einen kontinuierlichen digitalen Datenstrom auf Abruf von Ort A zu Ort B zu transportieren, nach wie vor bei allen beteiligten Parteien Interessenkonflikte. Der Anwender hat ausschließlich das Interesse einen Datenstrom beliebiger Kategorie kostengünstig, störungsfrei und sofort zu übertragen. Ein Netzbetreiber hat das Interesse, so viele Anwender wie möglich zufriedenzustellen. Ungeachtet desjenigen, der die Daten in das System einbringt, zeigt sich schon bei den erstgenannten Parteien ein unlösbares Problem. Der Anwender und der Netzbetreiber können einander nur näherkommen, wenn der Anwender z.B. die Daten komprimiert überträgt und der Netzbetreiber die Ressourcen erhöht. Derartige Entscheidungen müssen jedoch sehr genau überlegt werden, um sie wirtschaftlich und technisch vertretbar zu realisieren.

Ein weiterer Konflikt entsteht beim Anbieter der zu übertragenden Daten. Er fordert ein einfaches Einbringen der zu übertragenden Daten in das System. Außerdem möchte der Anbieter ebenso wie der Netzbetreiber möglichst viele Anwender zufriedenstellen und stößt dabei schnell an die Grenzen seiner verfügbaren Systemkapazitäten.

Die eben genannten Konflikte spitzen sich auf der Anwenderseite zu, wenn die Anwender untereinander in Konkurrenz treten; dies ist dann der Fall, wenn ihre Aktionen sich gegenseitig stören. Solche Effekte wirken sich gerade bei kontinuierlichen Datenströmen während der Darstellung in Echtzeit sehr störend aus.

Es gibt mittlerweile mehrere Ansätze, diesen Problemen entgegenzuwirken: z.B. soll der Einsatz neuer Protokolle und neuer Netztechnologien den Anwendern die Qualität der Übertragung zusichern, immer bessere Kompressionsverfahren sorgen für eine bessere Kanalausnutzung, eine dynamische Regelung der Bandbreite vermeidet extreme Störungen indem eine reduzierte Darstellungsqualität in Kauf genommen wird. Den genannten Ansätzen gemeinsam ist, daß sie das Problem transformieren, auf die maximale Anzahl der Anwender nehmen sie nur begrenzt Einfluß. Aus der direkten Konkurrenz aller Anwender untereinander wird eine ausschließende Konkurrenz oder aus abrupten Störungen werden durch Reduzieren der Informationen kontinuierliche, weniger auffällige Störungen.

Ein weiterer Ansatz, den begrenzten Systemressourcen zu begegnen, ist die intelligente Verteilung der Ressourcen. Für den Anbieter von abrufbaren Daten heißt dies, daß Repliken oder Teile seiner Daten auf weiteren Systemen bereitgestellt werden, wenn die Kapazität eines Systems erschöpft ist (verteilte Server-Technik). Für den Netzbetreiber bedeutet es, daß bei knappen Ressourcen Repliken oder Teile der zu übertragenden Daten in Netzsegmenten bereitgestellt werden, die sich möglichst nah beim Anwender befinden (Caching-Technik).

Ein derartig verteiltes On-Demand-System kann zwar eine sehr viel größere Menge an Anwendern zulassen, es erfordert jedoch einen erheblichen Verwaltungsaufwand, um den Bedürfnissen aller an der Übertragung beteiligten Parteien gerecht zu werden.

Motivation für die Entwicklung eines Media-Stream-Management-Systems

Die Abteilung „System Engineering“ der Firma SICAN integriert zur Zeit vermehrt On-Demand-Komponenten in ihre Systeme. Dabei wird der gesamte Produktionsprozeß vom Anbieter bis zum Anwender abgedeckt.

Ein typisches Szenario ist in Abbildung 1.1 dargestellt. Verschiedenste Video- und Audiobeiträge werden über die Video- bzw. Audiointerfaces in das System eingebracht. Redaktionsarbeitsplätze können das Material direkt sichten, lokal bzw. auf Servern mitschneiden und/oder bearbeiten. Mobile Arbeitsplätze können sich spontan an das System anschließen und Beiträge liefern, beziehen bzw. bearbeiten. Gleiches gilt für externe Arbeitsplätze, die sich beispielsweise über das Telefonnetz in das System integrieren. Fertige Beiträge können über spezielle Arbeitsplätze präsentiert werden usw.

Die Anforderungen an derartige Systeme, viele Stunden Video- bzw. Audiomaterial in Studioqualität zu speichern und verarbeiten zu können, zwingen zur Realisierung neuer verteilter Konzepte. Hierbei werden bei der SICAN GmbH Ansätze favorisiert, die eine Leistungssteigerung durch die intelligente Verteilung der Ressourcen ermöglichen. Der momentan verfolgte Ansatz mit dem Namen *Threaded Streams* zeichnet sich dadurch

aus, daß auch ein einzelner Video- bzw. Audiobeitrag selbst netzweit verteilt vorliegen kann, und daß die kleinste Einheit eines Teils des Beitrages frei wählbar ist. Eine ausgewogenere Auslastung der Ressourcen, bessere Leistungs- und Kapazitätsanpassung und höhere Robustheit sind einige der Argumente die für dieses Konzept sprechen.

Motivation für die Entwicklung eines Media-Stream-Management-Systems ist, allen beteiligten Komponenten eines solchen Systems die Verwaltung der verteilten Datenströme und deren benötigte Ressourcen sowie die Steuerung des Datenflusses anzubieten. Das zu entwickelnde System wird im folgenden kurz mit DMSM-System bzw. DMSMS (Distributed Media-Stream Management-System) bezeichnet. Die zu verwaltenden Datenströme verschiedener Medienklassen mit ihren möglichen inneren Strukturen werden allgemein durch den Begriff *Stream-Objekt* bezeichnet.

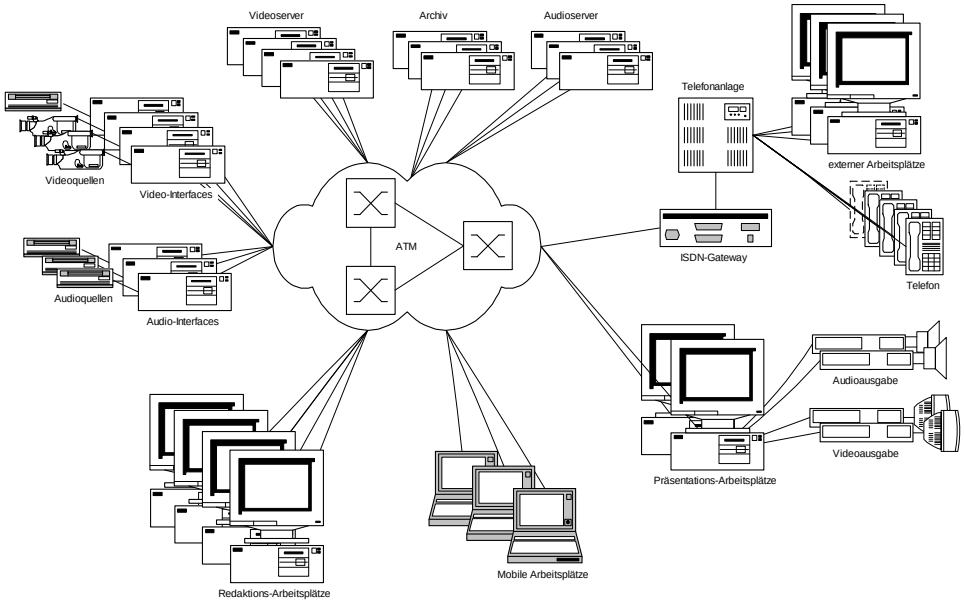

Abbildung 1.1 Szenario eines Video-, Audio-Redaktionssystems

Teil I

Systemanalyse

2 Zweckspezifikation

Zweck des DMSM-Systems ist es, Dienste für Applikationen bereitzustellen, die Stream-Objekte verwalten, sie in einem Netz lokalisieren, die für den Transport notwendigen Parameter bereitstellen, die benötigten Ressourcen verwalten und den Datenfluß steuern.

Das System soll so organisiert sein, daß Stream-Objekte jederzeit hinzugenommen und entfernt werden können, ohne zentrale Instanzen administrieren zu müssen, so daß die Verfügbarkeit eines Stream-Objektes nur abhängig ist von seiner Existenz, seinen Attributen und den für den Abruf nötigen Systemressourcen.

Stream-Objekte verwalten heißt auch, auf deren Lebenszyklus Einfluß zu nehmen, wobei der Lebenszyklus eines Stream-Objektes von seinem Entstehen bis zum Entfernen bzw. Archivieren reicht, sofern sich das Archiv außerhalb der Systemgrenzen des DMSM-Systems befindet.

Stream-Objekte, im Sinne des zu entwickelnden Systems, sind in erster Linie Informationsträger. Sie tragen die Informationen, die ein Anbieter netzweit zur Verfügung stellt. Eine detaillierte Betrachtung der Stream-Objekte erfolgt im Kapitel 5.2.2.

Mindestens drei Instanzen sind an der Übertragung eines Stream-Objektes beteiligt:

- Die Informationsquelle: Sie stellt das Stream-Objekt zur Verfügung.
- Das Netz: Es wird als Übertragungsmedium genutzt.
- Die Informationssenke: Sie empfängt das Stream-Objekt und verarbeitet es.

Allgemein läßt sich somit sagen: In einer Netzumgebung existieren Informationsquellen und Informationssenken, die mit Hilfe des DMSM-Systems verknüpft werden können. Auf der Grundlage der Verknüpfungen kann ein Informationsfluß, sowohl von den Quellen als auch von den Senken, initiiert werden. Dabei muß das DMSM-System nicht aktiv an der Übertragung beteiligt sein; es liefert dann ausschließlich die für die Übertragung nötigen Parameter und prüft die Verfügbarkeit der benötigten Ressourcen.

3 Systemumgebung

Die Systemumgebung, in die das DMSM-System integriert werden soll, sind Rechner, die an ein gemeinsames Datennetz angeschlossen sind. Das Datennetz kann heterogen sein und gleicht in seiner Beschaffenheit einem Intranet. Mit der Bezeichnung Intranet ist hier ein betriebsinternes Datennetz gemeint, das als abgeschlossene Einheit angesehen werden kann. Die vornehmlich eingesetzte Netztechnik ist ATM und Ethernet. Die angeschlossenen Rechner werden mit dem Betriebssystem Windows NT betrieben.

Wenn ein Stream-Objekt abgerufen wird, kommt ein Informationsfluß nur zwischen den angeschlossenen Rechnern zustande und gelangt immer über das Datennetz von den Quellen zu den Senken, wobei es durchaus zu einer Quelle mehrere Senken und zu einer Senke mehrere Quellen geben kann. Informationsquellen im Sinne des DMSM-Systems müssen nicht unbedingt den Ursprung der Informationen darstellen. Ein angeschlossener Rechner kann auch als Bindeglied fungieren, der die Informationen einer externen Quelle empfängt und – beispielsweise transformiert – wieder im Netz zur Verfügung stellt.

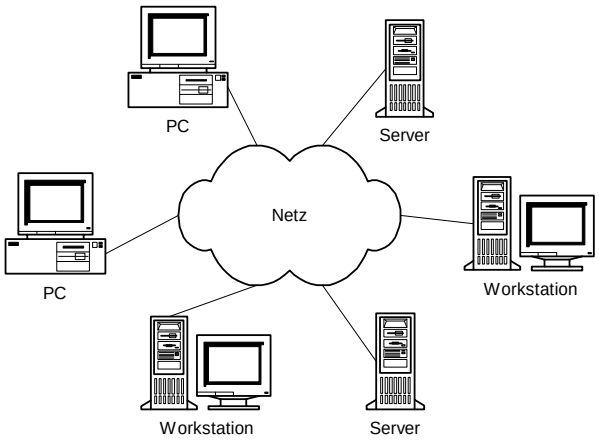

Abbildung 3.1 Systemumgebung des DMSM-Systems

Das System besitzt keine festgelegte Ausprägung. Komponenten können jederzeit hinzugefügt bzw. entfernt werden. Jederzeit entfernen bedeutet hier nicht nur das ord-

nungsgemäße Abschalten der Geräte, sondern auch den spontanen Ausfall eines Gerätes z.B. durch einen Defekt.

Aktivitäten von Applikationen, die Dienste des DMSM-System nutzen, führen zu Ereignissen innerhalb des DMSM-Systems und rufen dadurch Reaktionen hervor. Ebenso führen Aktivitäten des DMSM-Systems zu Ereignissen innerhalb der Applikationen. Auch hier werden Reaktionen hervorgerufen. Die meisten Applikationen werden von Benutzern spontan bedient, so daß ein Großteil der Aktivitäten spontan ist. Abbildung 3.2 zeigt ein Aktivitätsszenario mit vier Applikationen, von denen zwei über eine Benutzerschnittstelle verfügen.

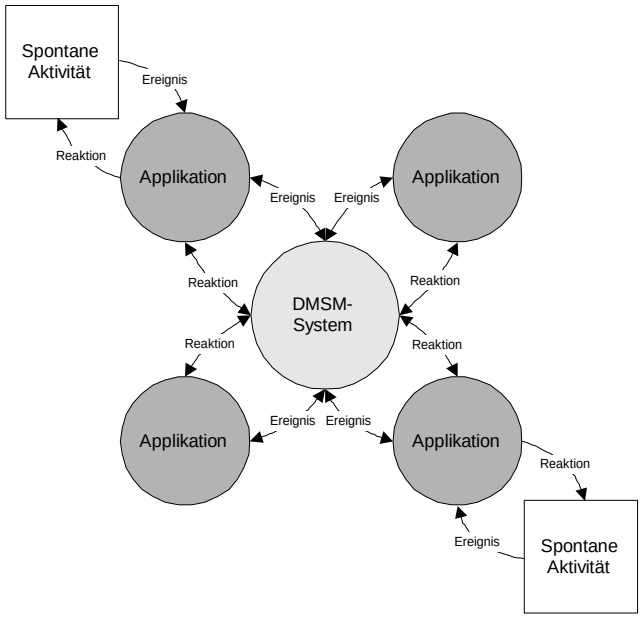

Abbildung 3.2 Aktivitäten des DMSM-Systems

4 Systemanforderungen

An Gesamtsysteme, in denen das DMSM-System zum Einsatz kommen soll, werden häufig hohe Sicherheitsanforderungen bzgl. des Ausfalls einzelner Komponenten gestellt. Das DMSM-System muß solche Fälle handhaben können und den Applikationen notwendige Informationen liefern.

Module, die an der Übertragung von Stream-Objekten beteiligt sind, müssen so gestaltet werden, daß sie leicht austauschbar sind und in der Lage sind, den Datentransport selbständig durchzuführen bzw. zu delegieren[1].

Ein Stream-Objekt kann auf *N* Systemkomponenten verteilt vorliegen. Das DMSM-System soll die Verwaltung solcher Stream-Objekte unterstützen und der Applikation eine einheitliche Sicht der Stream-Objekte liefern. Die gleiche Anforderung gilt auch für virtuelle Stream-Objekte. Virtuelle Stream-Objekte bestehen ausschließlich aus einer Folge von Referenzen auf reale Stream-Objekte.

Das System soll als verteiltes System ohne zentrale Verwaltungsinstanzen realisiert werden. Außerdem sollen die einzelnen Systemkomponenten ohne administrative Aufwände in das System integriert und entfernt werden können.

Damit die Arbeit im gegebenen zeitlichen Rahmen durchführbar ist, soll das Ressourcenmanagement nur analysiert, nicht aber entworfen bzw. implementiert werden. Das System soll aber eine nachträgliche Integration des Ressourcenmanagements vorsehen.

[1] In diesem Fall steuert das Modul ausschließlich den Datentransport. Der Datenpfad ist für das Modul nicht sichtbar.

5 Analyse

In allen Phasen der Entwicklung des DMSM-Systems wird die *Object Modeling Technique* (OMT)Coh95 angewandt. Auf Abweichungen von den Techniken der in [Coh95] spezifizierten Methoden wird an den entsprechenden Stellen hingewiesen.

Zunächst erfolgt die detaillierte Problembeschreibung, anschließend folgt die Objektmodellierung, die dynamische Modellierung und die funktionale Modellierung. Das funktionale Modell spezifiziert, was geschieht, das dynamische Modell spezifiziert, wann etwas geschieht, und das Objektmodell spezifiziert, in Bezug auf wen oder was etwas geschieht.

5.1 Problembeschreibung

In einem Netz existieren Informationsquellen, die Stream-Objekte anbieten, und Informationssenken, die Stream-Objekte konsumieren können. Es soll ein verteiltes Media-Stream-Management-System (DMSM-System) entwickelt werden, um Stream-Objekte zu verwalten und Beziehungen zwischen Quellen und Senken herstellen zu können. Applikationen können die Fähigkeiten des DMSM-Systems in Form von Diensten nutzen. Die Dienste stellen die DMSM-Systemkomponenten zur Verfügung. Applikationen, die das DMSM-System nutzen sind z.B. Streamserver (Live oder Aufzeichnung), Konferenzsysteme, Archivserver, Aufzeichnungsserver, Konverter.

Folgende Dienste soll das DMSM-Systems anbieten:

- Stream-Objekte können durch Angebote im Netz lokalisiert werden. Informationsquellen liefern Angebote entweder initiiert durch die Anfrage einer Informationssenke oder aus Eigeninitiative. Die Applikationen erhalten in den Angeboten Informationen über verfügbare Stream-Objekte bzgl. Ort, Format, Ressourcenanforderung und Zustand. Ob ein Stream-Objekt von einer Applikation lokalisiert werden kann,

ergibt sich aus den Attributen des Stream-Objektes selbst. Die Attribute werden von der Informationsquelle bestimmt. Führt eine Anfrage, z.B. durch replizierte Stream-Objekte, zu mehrdeutigen Antworten, ist die Wahl so zu treffen, daß die Ressourcen ausgewogen verteilt und Prioritäten höherer Ordnung berücksichtigt werden.

- Auf der Grundlage von Beziehungen kann ein Transport der Stream-Objekte erfolgen. Alle für den Transport notwendigen Parameter werden an die Applikationen geliefert. Applikationen sollen den Stream-Transport steuern können. Die Steuerung des Stream-Transportes soll polymorph bzgl. normaler, verteilter und virtueller Stream-Objekte sein. Steuerungen anderer Kategorien sollen ebenfalls durch das System ermöglicht werden können.
- Das Format eines Stream-Objektes soll nicht direkt über seine Verfügbarkeit entscheiden, vielmehr soll das DMSM-System alternative Verknüpfungen zwischen Systemkomponenten ermitteln, um das Stream-Objekt in das gewünschte Format zu transformieren. Ob eine Übertragung unter den von der Informationsquelle gestellten Anforderungen zustande kommt und wieviel Komponenten maximal an der Übertragung beteiligt werden dürfen, entscheiden die verfügbaren Ressourcen und die entstehenden Kosten.
- Die Ressourcen der DMSM-Systemkomponenten werden überwacht und Informationen über die Zustände der Systemkomponenten an die Applikationen geliefert. Ressourcen sollen gebucht werden können, damit eine kontinuierliche Übertragung eines verteilten bzw. virtuellen Stream-Objektes gewährleistet werden kann.
- Es sollen Dienste bereitgestellt werden, die Applikationen folgendes ermöglichen: Das Hinzufügen und Entfernen von Stream-Objekten zu registrieren, Repliken von Stream-Objekten zu erzeugen und Stream-Objekte zu archivieren.

5.2 Objektmodellierung

Das Objektmodell soll die statischen Strukturen des DMSM-Systems erfassen. Es zeigt Objekte im System, die Relationen zwischen den Objekten und die Attribute und Operationen, die jede Klasse von Objekten charakterisieren. Nach der Extraktion konkreter Klassen aus der Problembeschreibung werden diese detailliert analysiert. Oft wird es notwendig sein, dynamische wie funktionale Aspekte zu bedenken, um das Wesen der Objekte präzise erfassen zu können. Sie erheben aber nicht den Anspruch auf Vollständigkeit; eine ergiebige dynamische und funktionale Analyse findet in den entsprechenden Kapiteln statt.

Werden die Substantive der Problembeschreibung extrahiert und konkretisiert, ergeben sich folgende Klassen:

DMSM-System, DMSM-Systemkomponente, Stream-Objekt, Informationsquellen, Informationssenken, Netz, Beziehung, Applikation, Dienst, Anfrage, Angebot, Transport, Steuerung, Ressource

Aus den folgenden Kontexten ergeben sich weitere Klassen:

- ... Stream-Objekte können durch Angebote im Netz lokalisiert werden ...
  → *Konnexionsdienst*
- ... kann ein Transport der Stream-Objekte erfolgen ...
  → *Transportdienst*
- ... Applikationen sollen den Stream-Transport steuern können ...
  → *Steuerungsdienst*
- ... Ressourcen sollen gebucht werden können ...
  → *Ressourcendienst*
- ... Das Hinzufügen und Entfernen von Stream-Objekten zu registrieren ...
  → *Registraturdienst*
- ... Repliken von Stream-Objekten zu erzeugen ...
  → *Replikationsdienst*
- ... und Stream-Objekte zu archivieren ...
  → *Archivdienst*

5.2.1 Objektverzeichnis

DMSM-System	Das DMSM-System bildet das Gesamtsystem bestehend aus *N* DMSM-Systemkomponenten, die über ein Datennetz in Beziehung zueinander treten können.
DMSM-Systemkomponente	Eine DMSM-Systemkomponente bildet eine abgeschlossene Einheit, die einen oder mehrere Dienste im DMSM-System anbietet und/oder nutzt. Eine DMSM-Systemkomponente definiert sich ausschließlich aus der Fähigkeit, mit anderen DMSM-Systemkomponenten Beziehungen einzugehen. Darüber hinaus kann eine DMSM-Systemkomponente entweder eine Informationsquelle, eine Informationssenke oder beides sein. Es können durchaus mehrere DMSM-Systemkomponenten auf einem physikalischen Gerät realisiert sein.
Stream-Objekt	Ein Stream-Objekt enthält die zu übertragenden Informationen in einem bestimmten Format. Darüber hinaus sind dem Stream-Objekt Metainformationen über Ressourcenanforderung, Ursprung, Typ usw. beigefügt. Ein Stream-Objekt kann parallel oder sequentiell verteilt vorliegen. Eine Sonderform des Stream-Objektes ist das virtuelle Stream-Objekt, daß ausschließlich aus Referenzen auf reale Stream-Objekte besteht.

Ein Stream-Objekt kann Informationsquelle wie -senke gleichermaßen sein.

Informationsquellen Eine Informationsquelle sendet die zu übertragenden Informationen eines Stream-Objektes an eine oder mehrere DMSM-Systemkomponenten, die Informationssenken darstellen.

Informationssenken Eine Informationssenke empfängt die zu übertragenden Informationen eines Stream-Objektes von einem oder mehreren DMSM-Systemkomponenten, die Informationsquellen darstellen.

Netz Das Netz dient dem DMSM-System als Übertragungsmedium für die Informationen eines Stream-Objektes und als Kommunikationsmedium.

Beziehungen Eine Beziehung kann zwischen DMSM-Systemkomponenten hergestellt werden. Eine Beziehung kann dann zustande kommen, wenn das Format der zu übertragenden Informationen von beiden Systemkomponenten verarbeitet werden kann und wenn beide Systemkomponenten über mindestens ein gemeinsames Protokoll in einem gemeinsamen Netz verfügen. Beziehungen können von einer Informationsquelle als auch von einer Informationssenke initiiert werden.

Applikation Eine Applikation nutzt die von einer DMSM-Systemkomponente angebotenen Dienste. Solche Applikationen sind z.B. Streamserver (Live oder Aufzeichnung), Konferenzsysteme, Archivserver, Aufzeichnungsserver, Konverter.

Dienst Dienste werden von einer DMSM-Systemkomponente angeboten. Es gibt verschiedene Dienstkategorien, die auch voneinander abhängig sein können. So kann z.B. der Replikationsdienst den Konnexionsdienst und den Transportdienst benutzen. Ein Dienstobjekt erweitert die Fähigkeiten einer DMSM-Systemkomponente.

Anfrage Eine Anfrage enthält Informationen zu gewünschten Stream-Objekten. Sie ist an das gesamte DMSM-System gerichtet und provoziert Angebote von Seiten der einzelnen DMSM-Systemkomponenten. Eine Anfrage impliziert immer die Suche nach einer komplementären DMSM-Systemkomponente. Eine Informationssenke wird immer nach einer Informationsquelle fragen und umgekehrt.

Angebot Ein Angebot enthält Informationen zu Stream-Objekten, die eine DMSM-Systemkomponente zur Verfügung stellt. Ein Angebot, das eine DMSM-Systemkomponente äußert, wird im gesamten DMSM-System geäußert. Ein durch eine Anfra-

ge provoziertes Angebot wird hingegen nur der anfragenden DMSM-Systemkomponenten präsentiert.

Transport Der Transport bezeichnet die Übertragung eines Stream-Objektes. Für jeden Transport wird ein Transportobjekt angelegt, das die Übertragung eines Stream-Objektes darstellt. Dabei muß ein Transportobjekt nicht aktiv am Datentransport beteiligt sein, es repräsentiert dann lediglich die belegten Ressourcen und den Zustand der Übertragung. Transportobjekte können mit einem Termin versehen werden, so daß Ressourcen für einen Transport gebucht werden können. Für die Verwaltung der Transportressourcen steht der Ressourcendienst zur Verfügung.

Steuerung Mit Hilfe eines Steuerungsobjektes können während eines Stream-Transportes auf einfache Weise Steuerkommandos zwischen den DMSM-Systemkomponenten ausgetauscht werden. So kann z.B. ein Steuerungsobjekt auf den Transport des Stream-Objektes Einfluß nehmen. Steuerungsobjekte werden vom Steuerungsdienst verwaltet.

Ressource Eine Ressource im Sinne des DMSM-Systems kann z.B. sein: Speicherkapazität, Bandbreite, Prozessorleistung usw. Alle Ressourcen können Kosten erzeugen. Somit sind Kosten ein Attribut einer Ressource. Eine DMSM-Systemkomponente besitzt begrenzte Ressourcenkapazitäten, die vom Ressourcendienst verwaltet werden.

Konnexionsdienst Der Konnexionsdienst kann Anfragen und Angebote äußern und entgegennehmen. Eingehende Angebote werden den Anfragen zugeordnet, um Beziehungen zu erstellen. Die Applikation wird je nach von ihr geäußertem Interesse informiert.

Transportdienst Der Transportdienst verwaltet die Transportobjekte und interagiert mit der Applikation sowie mit dem Ressourcendienst bzgl. aller transportrelevanten Belange.

Steuerungsdienst Der Steuerungsdienst verwaltet alle Steuerungsobjekte und führt die, für die Steuerung relevante, Kommunikation mit der Applikation durch.

Ressourcendienst Mit Hilfe des Ressourcendienstes können Anfragen bzgl. verfügbarer Ressourcen gestellt werden. Dabei stehen einerseits die Zustände der einzelnen an einer Übertragung beteiligten Hardwarekomponenten zur Verfügung und andererseits alle von dem Transportdienst gestellten Ressourcenanforderungen. Der Ressourcendienst führt über sämtliche Ressourcenanforderungen Statistiken, die von anderen Diensten genutzt werden können.

Registraturdienst Der Registraturdienst dient dem Erfassen neuer Stream- und Dienstobjekte. Applikationen können sich über diesen Dienst beim DMSM-System anmelden, die DMSM-Systemkomponente bzgl. ihrer Anforderungen konfigurieren und das Einbinden neuer Objekte erfahren oder veranlassen.

Replikationsdienst Mit Hilfe des Replikationsdienstes können Repliken von Stream-Objekten auf DMSM-Systemkomponenten gleichen Diensttyps erzeugt werden. Damit kann einem Ressourcenmangel vorgebeugt werden, sofern die Replikation früh genug eingeleitet wird. Für eine Entscheidungshilfe kann die vom Ressourcendienst geführte Zugriffsstatistik benutzt werden. Der Replikationsdienst muß die für eine Replikation notwendigen Ressourcen früh genug buchen, um bei Ressourcenmangel genügend Reserven zu haben.

Archivdienst Der Archivdienst kann genutzt werden, um den Archivierungprozeß zu automatisieren. So kann auf Grundlage der Statistiken des Ressourcendienstes das Zugriffsverhalten auf Stream-Objekte ermittelt werden, um über die Archivierung eines Stream-Objektes zu entscheiden. Die Archivierung kann dann mit Methoden des Archivdienstes durchgeführt werden.

Die im Objektverzeichnis aufgeführten Dienste decken alle bisher geäußerten Anforderungen ab. Die Dienste sollen so in das System integriert werden, daß eine Erweiterung der Dienste einfach möglich ist und daß nicht benötigte Dienste entfernt werden können. Um eine DMSM-Systemkomponente in das gesamte DMSM-System zu integrieren, ist der Konnexionsdienst zwingend notwendig. Eine DMSM-Systemkomponente, die ausschließlich diesen Dienst implementiert kann z.B. eine Applikation zur Darstellung der Systemausprägung sein, Stream-Objekte selbst werden dabei nicht abgerufen.

An dieser Stelle ist noch nicht eindeutig geklärt, ob sich die Fähigkeiten einer DMSM-Systemkomponente in den Fähigkeiten der Dienste manifestieren oder ob die Applikation mit einbezogen werden soll. Im ersten Fall würde die Konfiguration der Dienste die Fähigkeiten implizieren, im zweiten Fall würde eine Applikation durch die Art und Weise, wie sie die Schnittstelle bedient, die Fähigkeiten implizieren. Die Entscheidung hierüber ist aber eine Entwurfentscheidung und wird zu einem späteren Zeitpunkt diskutiert. Anzumerken ist an dieser Stelle noch: Da mehrere Applikationen die gleichen Dienste benutzen können, wäre die erste Variante nur möglich, wenn die Applikationen ihren Bedarf an Dienstleistungen beim Start anmelden. Die Anmeldung würde dann beim Registraturdienst erfolgen, womit dann auch dieser Dienst bei der Integration einer DMSM-Systemkomponente in das gesamte DMSM-System zwingend notwendig ist.

Bevor einzelne Objekte genauer analysiert werden, soll in der folgenden Abbildung der erste Absatz der Problembeschreibung in einem Objektdiagramm abgebildet werden, um die globalen Zusammenhänge zu veranschaulichen:

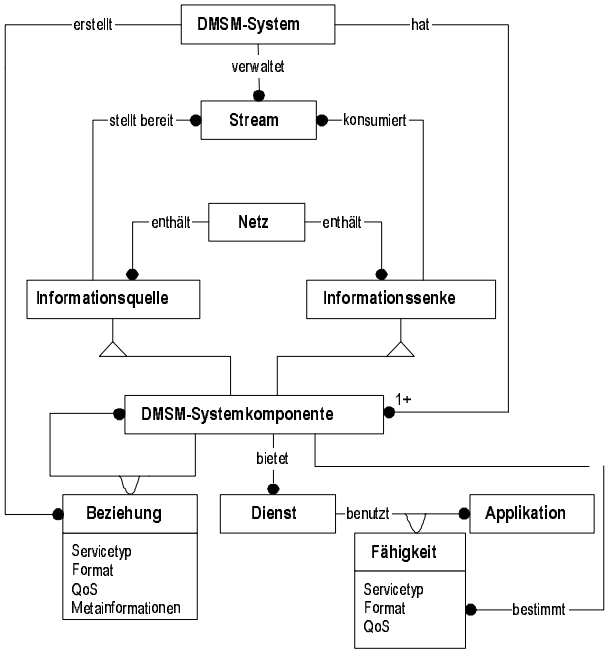

Abbildung 5.1 Problembeschreibung in OMT-Notation

Dem ersten Absatz der Problembeschreibung schließt eine Beschreibung der Dienste an, die einer Applikation zu Verfügung stehen sollen. Die Dienste, wie auch die Stream-Objekte sind die zentralen Objekte einer DMSM-Systemkomponente und sollen deshalb im folgenden genauer betrachtet werden.

Es werden in dieser Arbeit nur die essentiellen Dienstobjekte modelliert und entworfen. Dies sind der Konnexionsdienst, Transportdienst, Steuerungsdienst, Ressourcendienst und der Registraturdienst.

Das Modellieren und der Entwurf der Dienstobjekte Replikationsdienst und Archivdienst werden im Rahmen dieser Arbeit nicht durchgeführt. Sie sind Hilfsdienste, die die Funktionalität des Gesamtsystems nur durch Nutzung der essentiellen Dienstobjekte erweitern und können bei Bedarf später leicht hinzugefügt werden.

5.2.2 Stream-Objekte

Ein Stream-Objekt ist im Sinne des DMSM-Systems ein Informationsträger. Es enthält neben den eigentlichen Informationen, die durch das Netz übertragen werden sollen, Metainformationen bzgl. Format, Ressourceanforderungen, Typ und Inhalt. Ein reales Stream-Objekt kann z.B. ein Videofilm in Form einer Datei auf der Festplatte oder eine Live-Videosequenz sein, es kann aber auch eine Informationssenke sein z.B. in Form eines Aufnahme-Stream-Objektes, das Videosequenzen auf eine Festplatte aufzeichnen kann. Die Richtung des Datenflusses spielt bei Stream-Objekten des DMSM-Systems eine untergeordnete Rolle.

Klassifikation von Stream-Objekten

Stream-Objekte können parallel oder sequentiell verteilt vorliegen. Das bedeutet, daß ein Stream-Objekt auf *N=1, 2, ...* Systemkomponenten verteilt ist. Der gesamte Informationsgehalt *I* eines Streams ergibt sich dann durch Summieren des Informationsgehaltes I_i der *N* Teil-Streams:

$$I = \sum_{i=1}^{N} I_i$$

Ein Stream ist sequentiell verteilt, wenn zum Zeitpunkt *T* die Informationen von einer der N Systemkomponenten geliefert bzw. empfangen wird, so daß für den Informationsfluß gilt:

$$\left.\frac{dI(t)}{dt}\right|_T = \left.\frac{dI_i(t)}{dt}\right|_T$$

Ein Stream ist parallel verteilt, wenn zum Zeitpunkt *T* die Informationen von *N* verschiedenen Systemkomponenten geliefert bzw. empfangen werden, so daß für den Informationsfluß gilt:

$$\left.\frac{dI(t)}{dt}\right|_T = \sum_{i=1}^{N} \left.\frac{dI_i(t)}{dt}\right|_T$$

Die oben genannten Stream-Objekte haben gemeinsam, daß sie real existent sind, sie selbst somit Träger der Informationen sind, die mit ihnen assoziiert werden. Eine zweite Kategorie von Stream-Objekten stellen die virtuellen Stream-Objekte [Ein97] dar. Es handelt sich dabei um Referenzlisten, deren Listeneinträge ausschließlich reale Stream-Objekte referenzieren.

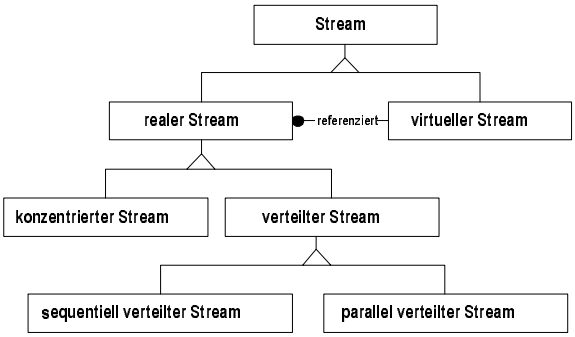

Abbildung 5.2 Vererbungshierarchie der Stream-Objekte

Problematik verteilter Stream-Objekte

Ein Problem, das sich bei verteilten Stream-Objekten in Bezug auf die Dynamik des Gesamtsystems stellt ist, daß jeder Teil-Stream Systemressourcen beansprucht, die ihm nicht jederzeit zugesichert werden können. Außerdem besteht die Möglichkeit z.B. beim Ausfall einer Systemkomponente, daß ein Teil-Stream nicht mehr im Gesamtsystem verfügbar ist. In einer solchen Situation stellt sich die Frage, ob es sinnvoll ist, alle ver-

fügbaren Teil-Streams zu übertragen. Die Frage läßt sich nur durch eine genauere Betrachtung der Auswirkungen bei Wegfall eines oder mehrerer Teil-Streams beantworten.

Ist ein sequentiell verteilter Stream auf *N* Systemkomponenten verteilt und fällt ein Teil-Stream weg, wird bei der Übertragung das *n*-te Teilstück fehlen. Wäre der betroffene Stream z.B. eine Video-Audiosequenz, würde die Wiedergabe für die Dauer des *n*-ten Teilstückes pausieren, sofern die Wiedergabe Timestamp-gesteuert erfolgt. Nun ist es davon abhängig, wie groß das fehlende Teilstück ist und welcher Inhalt betroffen ist. Unter diesen Umständen kann keine klare Aussage über die Auswirkungen getroffen werden.

Bei einem parallel verteilten Stream verhält es sich ähnlich. Fällt ein Teil-Stream aus und handelt es sich ebenfalls um eine Video-Audiosequenz, bei der die Audiospur und Videospur auf verschiedene Systemkomponenten verteilt wurden, fehlt entweder die Audiospur oder die Videospur. Die Auswirkungen sind auch hier nicht absehbar, da der Inhalt des betroffenen Mediums für die Auswirkungen ausschlaggebend ist.

Betrachtet man die getroffenen Aussagen, zeigt sich, daß die Inhalte der weggefallenen Teil-Streams die Auswirkungen bestimmen. Daraus folgt, daß die Redundanz des weggefallenen Teil-Streams bezogen auf den Gesamt-Stream Rückschlüsse auf die Auswirkungen zuläßt.

Am Beispiel einer Videosequenz, bei der die zeitliche Redundanz nicht reduziert wurde (keine Interframe-Komprimierung), kann gezeigt werden, daß die Bedingung für eine hohe Redundanz pro Teil-Stream erreicht werden kann: Angenommen, ein Stream wird zyklisch sequentiell auf *N* Systemkomponenten verteilt, wobei jeder Teil-Stream die konstante Länge *T* besitzt, und angenommen die Zeit *T* wird so klein gewählt, daß nur noch ein Videobild in einem Teil-Stream Platz findet, so entartet die Sequenz in eine Sammlung von Einzelbildern verteilt auf *N* Systemkomponenten. Faßt man nun die Einzelbilder auf jeweils einer Systemkomponente zu einem neuen Stream zusammen, erhält man wieder *N* Teil-Streams, bei denen jeder Teil-Stream jedes *N*-te Bild der ursprünglichen Sequenz enthält, mit einem Versatz von *v* Bildern bzgl. der anderen *N-1* Teil-Streams. Bei der Wiedergabe der Sequenz würden die Bilder zyklisch abgeholt werden und wieder zu einem Stream zusammengesetzt werden. Fällt nun ein Teil-Stream weg, würde jedes *N*-te Bild fehlen, was in vielen Fällen akzeptiert werden kann. Übertragen auf Audiodaten ergibt sich ein ähnlicher Effekt. Die kleinste Einheit der Partitionierung wäre in diesem Fall ein Sample. Der Wegfall einer Systemkomponente hätte dann eine Minderung des Frequenzganges und des Signal-Rauschabstandes zur Folge, sofern bei fehlenden Samples interpoliert wird.

Das Beispiel zeigt sehr deutlich, daß es unter Umständen sinnvoll ist, trotz Wegfall von Teil-Streams die Übertragung durchzuführen bzw. fortzuführen. Der Effekt, daß die Leistungsfähigkeit eines so realisierten Systems proportional zur Komponentenzahl wächst und hohen Sicherheitsanforderungen genügen kann, rechtfertigt den Einsatz eines solchen Verfahrens.

Wendet man das im Beispiel geschilderte Verfahren an, entsteht ein quasi parallel verteilter Stream. Ein Stream ist quasi parallel verteilt, wenn zwischen den Zeitpunkten *T*

und $T+\Delta t$ die Informationen von N verschiedenen Systemkomponenten geliefert werden, so daß für den Informationsfluß gilt:

$$\left.\frac{I(t+\Delta t)-I(t)}{\Delta t}\right|_T = \sum_{i=1}^{N} \left.\frac{I_i(t+\Delta t)-I_i(t)}{\Delta t}\right|_T$$

und $\Delta t << Tg$, wobei Tg die Wiedergabezeit des gesamten Streams meint.

Notwendige und optionale Merkmale eines Teil-Streams

Die vorangegangene Diskussion gilt uneingeschränkt auch für den Fall der Aufzeichnung und zeigt den Sinn einer Übertragung bei fehlenden Teil-Streams. Daraus ergeben sich die folgenden Forderungen an die beigefügten Metainformationen eines Teil-Streams:

- Ein Teil-Stream darf keine Abhängigkeiten von anderen Teil-Streams aufweisen.
- Ein Teil-Stream muß seinen Typ, sein Format, seine benötigten Ressourcen und seine Synchronisation im Gesamt-Stream selbst repräsentieren.
- Alle Teil-Streams müssen ein Merkmal enthalten, daß ihre Zusammengehörigkeit auszeichnet.

Gesucht ist nun eine einfach zu handhabende Datenstruktur, die bei minimalem Verwaltungsaufwand die Anforderungen erfüllt und alle möglichen Kombinationen von Stream-Klassen abdeckt. Ein Überblick über die Struktur möglicher Stream-Klassen und deren Kombinationen gibt Abbildung 5.3.

Eine Gruppierung von Teil-Streams kann über eine im Namensraum eindeutige ID erreicht werden. Hierzu eignet sich ein sogenannter UUID (universally unique identifier). Ein solcher UUID kann bei der Erstellung von Stream-Objekten einfach generiert werden und ist mit ausreichender Wahrscheinlichkeit weltweit eindeutig. Alle einem Gesamt-Stream zugehörigen Teil-Streams besitzen den selben UUID.

Um die Position eines Teil-Streams innerhalb des Gesamt-Stream eindeutig bestimmen zu können, sind zwei weitere Parameter notwendig. In Abbildung 5.3 ist dies die Nummer der Spur und die Sequenznummer. Die Nummer der Spur kann als solche direkt eingesetzt werden. Die Sequenznummer ist überflüssig, da sie durch den Startzeitpunkt im Gesamt-Stream impliziert wird.

Die Zugehörigkeit und Position eines Teil-Streams definiert sich somit aus dem Parametertripel (*UUID, Spur, Startzeitpunkt*). Um beurteilen zu können, ob Teilsequenzen fehlen, muß jedem Teil-Stream die gesamte Anzahl der Spuren und die gesamte Spielzeit bekannt gemacht werden. Es kann dann leicht bestimmt werden, ob das Fehlen von Teil-Streams toleriert werden kann.

Für einen einfachen nicht linearen Zugriff innerhalb des Gesamt-Streams und für eine einfache graphische Darstellung der Stream-Struktur ist es erforderlich, auch die Spielzeit eines Teil-Streams zu kennen. Außerdem sollte über Flags angezeigt werden können ob ein Überlappen oder eine unvollständige Überdeckung innerhalb einer Spur zulässig ist.

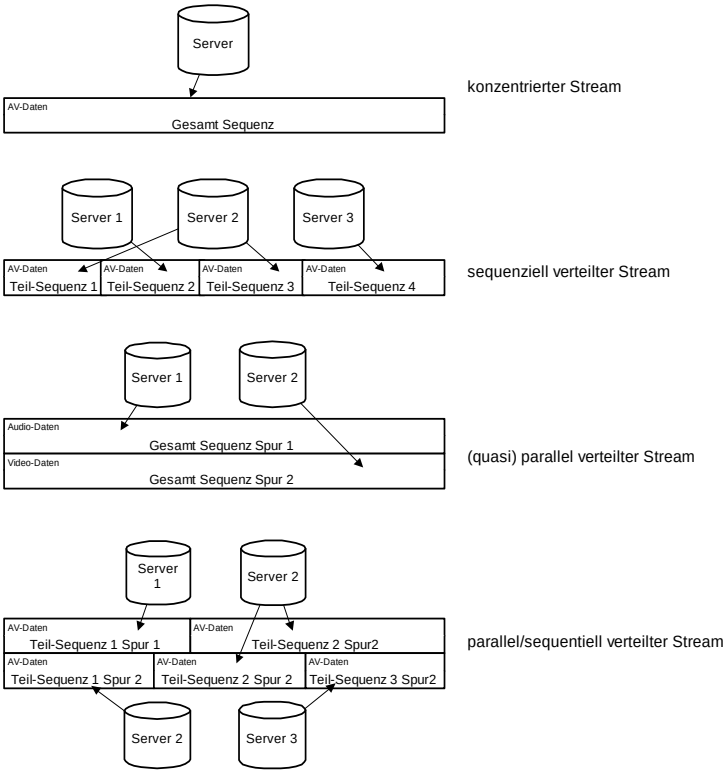

Abbildung 5.3 Stream-Klassen und deren Kombinationen

Eine weitere Forderung ist, daß jeder Teil-Stream seinen Typ, sein Format und seine benötigten Ressourcen spezifiziert. Der Typ wird durch die Medienklasse des Streams bestimmt wie z.B. Audio, Video oder Text. Das Format impliziert die Codierung und das Dateiformat. Für die Beschreibung des Typs und des Formates eignet sich die Konvention der *Media Types* (ehemals *MIME Types* [RFC1341]), die von der *Internet Assigned Numbers Authority* (IANA) zentral verwaltet werden [RFC1590]. Die Ressourcenanforderungen können z.B. durch die *Flow-Spec*-Datenstruktur [RFC1363] beschrieben werden.

Neben den bisher genannten Metainformationen, die für eine effiziente Verwaltung notwendig sind, sollten weitere Informationen angefügt werden. So wären z.B. Attribute denkbar, die es gestatten, bei einem parallel verteilten Video-/Audio-Stream die Audio-Spur nach Vorgabe einer Sprache zu wählen. Weitere Attribute wären z.B. Name, Copyright, Erstellungszeitpunkt, Lebenszeit, Zugriffsrechte, Inhaltsbeschreibung, Verweise, suchbare Begriffe etc. Alle Attribute dieser Kategorie werden unter dem Datum *optionale Beschreibung* zusammengefaßt.

Ein Teil-Stream wird damit durch das folgende Datentupel beschrieben:

Teil-Stream = (*UUID, Spuranzahl, Spur, Startzeit, Endzeit, Gesamtspielzeit, Flags, Media Type, Flow Spec, optionale Beschreibung*)

Eine genauere Untersuchung der einzelnen Datenelemente und deren Datentypen erfolgt erst in der Entwurfsphase. An dieser Stelle wird auch bewußt auf die Darstellung der Assoziationsmethoden verzichtet, mit denen die Metadaten und die eigentlichen Stream-Daten in Beziehung gebracht werden, da eine Entscheidung für oder wider spezieller Methoden ebenfalls eine Entwurfsentscheidung ist. Es wird zunächst lediglich vorausgesetzt, daß eine Assoziation besteht. Damit ergibt sich folgendes Klassenmodell für ein DM-Stream:

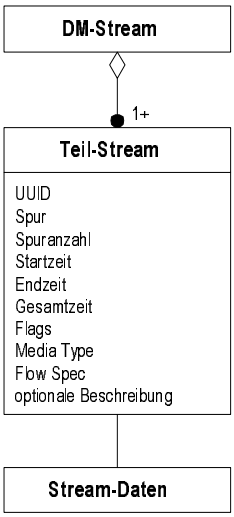

Abbildung 5.4 Objektmodell eines DM-Streams

Das dargestellte Objektmodell eignet sich ebenfalls für die Beschreibung von virtuellen Stream-Objekten. Die Aggregation ist bei realen DM-Streams nur implizit gegebenen, nämlich durch die Existenz der Teil-Streams, während bei virtuellen DM-Streams eine explizite Aggregation besteht. Das heißt, es existiert eine Instanz eines DM-Streams, die eine oder mehrere Teil-Stream-Instanzen enthält, wobei die Assoziation mit den Stream-Daten nicht direkt erfolgt, sondern indirekt durch die UUID.

5.2.3 Konnexionsdienst

Damit eine sinnvolle Verbindung von DMSMS-Komponenten zustande kommen kann und darüber ein Datenfluß eingeleitet werden kann, müssen neben der physikalischen Kopplung weitere Voraussetzungen geschaffen werden. Aufgabe des Konnexionsdienstes ist es, diese Voraussetzungen zu schaffen, wie beispielsweise die Integration von Komponenten in das DMSM-System.

Komponentenfähigkeiten

Zunächst muß eine Komponente genau wissen, welche *Media Types* (MT) sie verarbeiten kann, welche Protokolle (PR) auf den physikalischen Medien dafür zur Verfügung stehen und ob sie als Senke und/oder Quelle fungieren soll. Zusammengefaßt wird dieses a priori Wissen als Komponentenfähigkeiten (KF) bezeichnet. Die i-te Fähigkeit der Komponente X kann durch das folgende Datentupel beschrieben werden:

$KF_i(X) = (MT_i(X), PR(X \mid MT_i(X)), RI_i(X))$

wobei

$MT_i(X)$	*Media Type* der Fähigkeit i der Komponente X
$PR(X) = \{PR_1(X), PR_2(X), \ldots, PR_l(X)\}$	Menge der Protokolle der Komponente X
$PR(X \mid MT_i(X)) \subseteq P(X)$	Menge der Protokolle der Komponente X, die bei der Nutzung des *Media Type* $MT_i(X)$ zur Verfügung stehen
$RI_i(X) \in \{IN, OUT\}$	Datenrichtung der Fähigkeit i aus Sicht der Komponente X

Damit ergibt sich für die Menge aller Fähigkeiten einer Komponente X:

$KF(X) = \{KF_1(X), KF_2(X), \ldots, KF_k(X)\}$

Mit Hilfe des Wissens über die eigenen Fähigkeiten einer Komponente kann der Konnexionsdienst nun Beziehungen zu verträglichen Komponenten herstellen. Es wird vorausgesetzt, daß jede im DMSM-System befindliche Komponente durch Präsentation seiner Fähigkeiten bekannt ist. Auf das Problem *Bekanntmachen* wird während der dynamischen Modellierung genauer eingegangen.

Komponentenverträglichkeit

Beziehungen sind also nur dann möglich, wenn Komponenten verträglich sind. Die Verträglichkeit von Komponenten wird wie folgt definiert:

$$V(A,B) = \begin{cases} \text{WAHR} \quad \textit{wenn} \quad \begin{cases} MT(A,B) = MT(A) \cap MT(B) \neq \varnothing \quad \textit{und} \quad A \neq B \quad \textit{und} \\ PR(A,B) = PR(A \mid MT(A,B)) \cap PR(B \mid MT(A,B)) \neq \varnothing \end{cases} \\ \text{FALSCH} \quad \textit{sonst} \end{cases}$$

wobei

$MT(X) = \{MT_1(X), MT_2(X), \ldots, MT_k(X)\}$	Menge der *Media Types* der Komponente X
$PR(X) = \{PR_1(X), PR_2(X), \ldots, PR_L(X)\}$	Menge der Protokolle der Komponente X
$PR(X \mid MT_i(X)) \subseteq P(X)$	Menge der Protokolle der Komponente X, die bei der Nutzung des *Media Type* $MT_i(X)$ zur Verfügung stehen.

Beziehungen sind demnach zwischen äquivalenten bzw. komplementären Komponenten möglich. Eine Äquivalenzbeziehung ist dann gegeben, wenn für die verträgliche Fähigkeit i gilt, daß die Datenrichtungen der Komponenten A und B übereinstimmen:

$RI_i(A) = RI_i(B) \neq \varnothing$ und $A \neq B$

Eine Komplementärbeziehung ist gegeben, wenn für die verträgliche Fähigkeit i gilt, daß die Datenrichtungen der Komponenten A und B entgegengesetzt sind:

$RI_i(A) \neq RI_i(B)$ mit $RI_i(A) \neq \emptyset$ und $RI_i(B) \neq \emptyset$ und $A \neq B$

Ein Datenfluß kann nur auf Basis einer Komplementärbeziehung zustande kommen, so daß es den Anschein hat, ausschließlich Komplementärbeziehungen seien sinnvoll. Wenn es aber darum geht, durch Äquivalenzbeziehungen Komponenten zu einer Einheit (z.B. verteilter Videoserver) zu gruppieren, oder bei einem Ressourcenmangel alternative Komponenten mit gleichen Eigenschaften zu finden, werden Äquivalenzbeziehungen ebenfalls sinnvoll.

In Abbildung 5.5 sind als Beispiel die möglichen Beziehungen zwischen vier verträglichen DMSMS-Komponenten dargestellt.

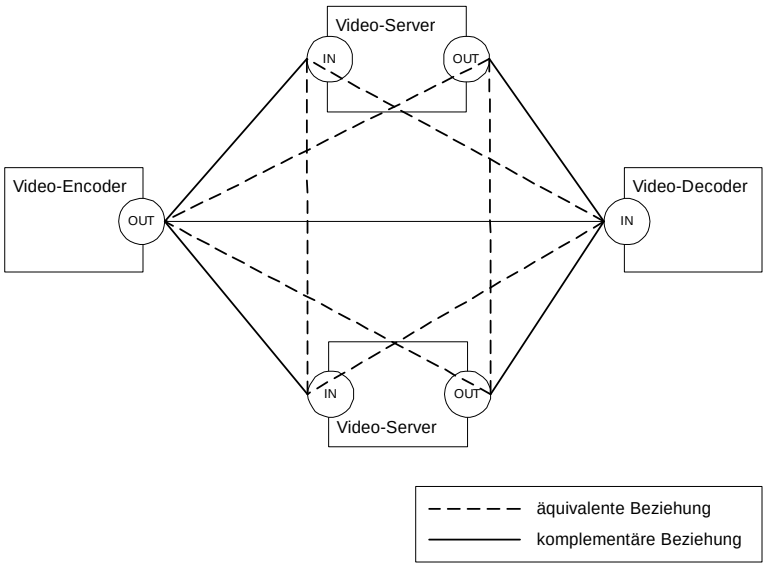

Abbildung 5.5 Mögliche Beziehungen verträglicher Komponenten

Der Videoencoder hat durch die Äquivalenzbeziehungen zwischen den Eingängen der Videoserver und dem Videodecoder die Möglichkeit einer direkten Übertragung zum Videodecoder, oder er kann ein konzentriertes bzw. einen parallel verteiltes Video-Stream-Objekt erzeugen. Ebenso hat der Video-Decoder die Möglichkeit zwischen dem Stream-Objekten des Video-Decoder oder den Stream-Objekten der Videoserver zu wählen. Auch hier können die Videoserver zu einem verteilten Videoserver gruppiert werden, bedingt durch die Äquivalenz der Videoserver-Ausgänge.

Konnexionslayer

Das Beispiel zeigt, daß sich innerhalb des DMSM-Systems durch die Beziehungen bestimmte Topologien bilden. Solche Topologien sind einzig von den Fähigkeiten der Komponente abhängig, sie werden im folgenden durch den Begriff Konnexionslayer (KL) bezeichnet. Brücken zwischen verschiedenen Konnexionslayer sind nur durch Konverter realisierbar, sie sind dann Teil mehr als eines Layers.

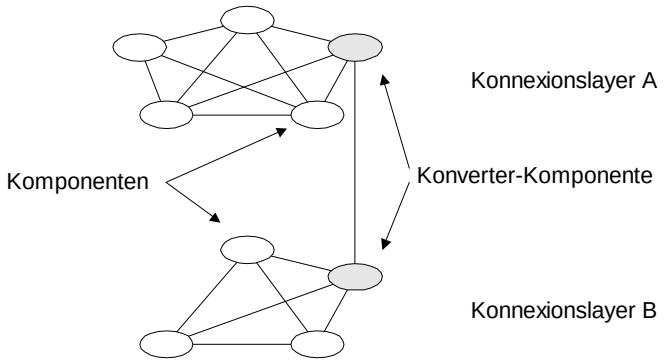

Abbildung 5.6 Beispiel zweier Konnexionslayer und ihre Verbindung

Komponentenfunktionalität

Für sinnvolle Verbindungen im Sinne der Datenübertragung zwischen DMSMS-Komponenten reicht das Wissen über die Fähigkeiten aus, nicht aber im Sinne der Anwendung, es muß auch die Funktionalität (FU) bekannt sein, die eine DMSMS-Komponente mit Hilfe ihrer Fähigkeiten bereitstellt. Realisiert werden die Funktionalitäten durch die Anwendungen, die die DMSMS-Dienste nutzen.

$FU(X) = \{FU_1(X), FU_2(X), \ldots, FU_k(X)\}$	Menge der Funktionalitäten der Komponente X
$FU_i(X) \in \{SERVER, \ldots, KONVERTER\}$	Funktionalität der Komponente X realisiert durch die Anwendung i

Die Trennung von Funktionalität und Fähigkeit ist sinnvoll, da eine Funktionalität immer mit einem bestimmten Typ von Anwendungen assoziiert wird, hingegen wird mit einer Fähigkeit eine Gruppe verschiedener Typen von Anwendungen assoziiert, so daß alternative Wege leichter gefunden werden können. Am Beispiel des Konverters wird dies deutlich: Sucht eine Anwendung nach Stream-Objekten auf Videoservern, kann ein Konverter eine transparente Sicht in einen anderen Konnexionslayer bieten, die bei der Forderung nach der Funktionalität SERVER verborgen bliebe.

Die bisher beschriebene Aufgabe des Konnexionsdienstes, die Präsentation der eigenen Fähigkeiten und das Bilden der Konnexionslayer, geschieht unmittelbar nachdem sich eine Anwendung beim Konnexionsdienst angemeldet hat. Der Konnexionsdienst stellt für die Anwendungen quasi eine Datenbank dar, die sämtliche Informationen über verträgliche Komponenten bereitstellt. Der Vorteil dieser Architektur ist, daß die Datenbank jederzeit der aktuellen Konfiguration des gesamten Systems entspricht. Über etwaige Änderungen der Systemausprägung können Anwendungen während ihrer Laufzeit informiert werden.

Anfragen und Angebote

Eine weitere Aufgabe des Konnexionsdienstes ist das Entgegennehmen und Äußern von Angeboten und Anfragen bzgl. Stream-Objekten. Angebote werden geäußert, wenn z.B. ein neues Stream-Objekt in einer DMSM-Systemkomponente integriert wird oder wenn

das Angebot durch eine Anfrage provoziert wurde. Eine Anfrage kann über den Konnexionsdienst gestellt werden. Der Konnexionsdienst sorgt in einem solchen Fall für die Veröffentlichung innerhalb der einzelnen Konnexionslayer in Abhängigkeit der gestellten Anfrage.

Im Grunde genommen hat der Konnexionsdienst bzgl. der Anfragen und Angebote nur eine vermittelnde Funktion. Dies hat die Konsequenz, daß bestimmte Anforderung an die Architektur der Applikation gestellt werden müssen. Da eine Anfrage unpräzise erfolgen kann, also eine unbestimmte Anzahl von Antworten der Anfrage entsprechen können, ist die Zeitspanne für das Eintreffen der Antworten nur schwer voraussagbar. Die Applikation müßte in diesem Fall asynchron bei jedem Eintreffen einer Antwort benachrichtigt werden. Dieser funktionale Aspekt wird während der dynamischen Modellierung genauer betrachtet. Was jedoch an dieser Stelle herausgearbeitet werden muß, ist die Form von Anfragen und Angeboten.

Ziel einer Anfrage sollte es sein, Stream-Objekte zu lokalisieren, die den in der Anfrage formulierten Anforderungen entsprechen. Eine Anfrage kann nur beantwortet werden, wenn eine Rückmeldeadresse des Anfragenden bekannt ist. Diese kann mit der Anfrage selbst geliefert werden. Alle Angebote können dann an die angegebene Rückmeldeadresse gesandt werden. Wird eine Anfrage befriedigend beantwortet, muß es möglich sein, mit Hilfe der Informationen aus der Anfrage und den Angeboten einen Datentransport direkt bzw. indirekt zu initiieren.

Für einen Datentransport ist gewöhnlich eine dezidierte Verbindung notwendig. Ein allgemeines Vorgehen für den Aufbau einer Transportverbindung und den Datentransport selbst ist nicht gegeben, so daß für jeden Konnexionslayer a priori Wissen vorliegen muß, das die Methodik für einen Verbindungsaufbau bzw. den Datentransport repräsentiert. Mögliche Formen und Orte der Wissensrepräsentation werden im Abschnitt *Transportdienst* erörtert.

Da ein durch eine Anfrage provoziertes Angebot nicht den Verbindungswunsch impliziert, reicht es aus, daß der Anfragende eine Adresse erfährt, an die er sich wendet, wenn eine Transportverbindung aufgebaut werden soll. Eine solche Adresse kann einem Angebot beigefügt werden.

An eine Anfrage wird die Forderung gestellt, daß eine Zuordnung zwischen gestellter Anfrage und eingehenden Angeboten durchführbar ist. Eine Zuordnung wäre mit der Eindeutigkeit der Rückmeldeadresse gegeben. Wenn aber durch eine Beschränkung des Adressraumes die Eindeutigkeit einer Rückmeldeadresse über einen längeren Zeitraum nicht gewährleistet werden kann, besteht die Möglichkeit, jeder Anfrage eine eindeutige Identifikation beizufügen, die bei jedem Angebot einen Bezug zu einer Anfrage darstellt. Eine ausgezeichnete Identifikation signalisiert dann ein Angebot aus Eigeninitiative.

Damit ergibt sich das folgende Datentupel für eine Anfrage:

Anfrage = ([ID], Suchbegriffe, Suchkriterien, Adresse)

Angebote, die aus Eigeninitiative versendet werden, entsprechen den Angeboten, die durch Anfragen provoziert werden. Ein Unterschied ergibt sich nur bei der Anzahl der

Empfänger; Angebote, die aus Eigeninitiative versandt werden, sind an alle Komponenten innerhalb eines Konnexionslayers adressiert.

Mit der Forderung, daß die Angebote alle notwendigen Daten liefern müssen, um auf die Struktur verteilter Stream-Objekte zu schließen, ergibt sich schließlich folgendes Datentupel:

Angebot = ([ID], Stream-Attribute, Adresse)

Die Stream-Attribute sind eine Untermenge der Teil-Stream-Attribute, zu ihnen gehören: *UUID, Spuranzahl, Spur, Startzeit, Endzeit, Gesamtspielzeit, Flow Spec, optionale Beschreibung*. Das Attribut *Media-Type* wäre an dieser Stelle redundant, da es durch den Konnexionslayer gegeben ist. Das Sammelsurium an Daten, das unter dem Oberbegriff *optionale Beschreibung* zusammen gefaßt wird, hat nicht unbedingt dieselbe Form wie das Original des Teil-Streams; die genaue Deklaration erfolgt während des Entwurfs.

War eine Objektsuche erfolgreich, impliziert dies gleichzeitig, daß eine dezidierte Verbindung zustande kommen kann, da eine Anfrage nur in einem Konnexionslayer gestellt werden kann und dieser aus verträglichen Komponenten besteht und Verträglichkeit laut Definition ein Protokoll auf einem gemeinsamen Netz fordert. Für den Aufbau einer Transportverbindung und die Durchführung der Datenübertragung ist der Transportdienst verantwortlich, der im nächsten Abschnitt behandelt wird.

Abbildung 5.7 zeigt das Analyseergebnis als Objektmodell in OMT-Notation.

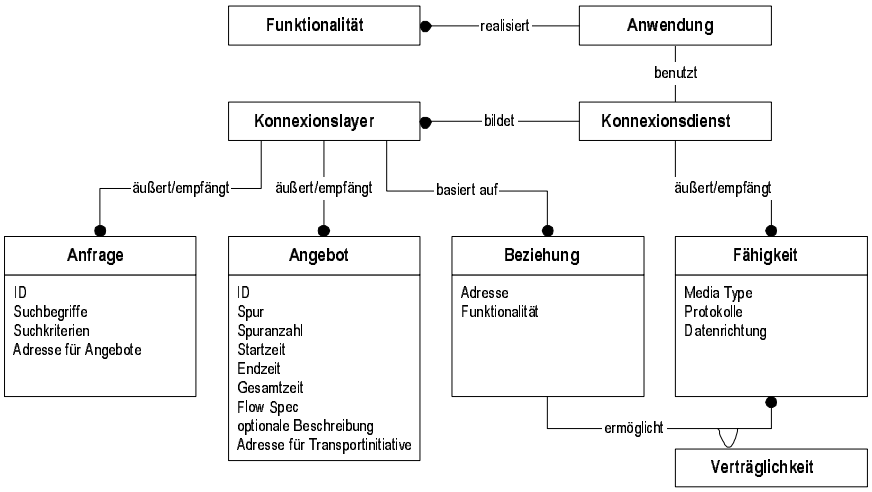

Abbildung 5.7 Objektmodell des Konnexionsdienstes in OMT-Notation

5.2.4 Transportdienst

Wurde mit Hilfe des Konnexionsdienstes ein Stream-Objekt gefunden, dessen Daten transportiert werden sollen, ist gewöhnlich eine dezidierte Verbindung notwendig. Ein allgemeines Vorgehen für den Aufbau einer Transportverbindung und den Datentransport selbst ist nicht gegeben, so daß für jeden Konnexionslayer a priori Wissen vorlie-

gen muß, das die Methodik für einen Verbindungsaufbau bzw. den Datentransport repräsentiert.

Transportobjekte

Nun stellt sich die Frage, wo das nötige a priori Wissen über die Methodik des Verbindungsaufbaus und den Datentransport untergebracht werden soll und kann. Das Wissen über den Verbindungsaufbau könnte beispielsweise in Transportobjekten gekapselt und im Transportdienst plaziert werden. Durch die Vielzahl der existierenden Netztechnologien müßte in diesem Fall das Wissen auf einfachste Weise erweitert bzw. ausgetauscht werden können.

Ein weiterer Ort, an dem das a priori Wissen über den Verbindungsaufbau untergebracht werden könnte, ist die Anwendung selbst. Dies ist aber nicht erstrebenswert, da dieses Wissen nicht an andere Anwendungen weitergegeben werden kann. Für Spezialfälle wäre es denkbar, ein Transportobjekt passiv auszulegen. Ein passives Transportobjekt wäre weder am Verbindungsaufbau noch am Datentransport direkt beteiligt. Es böte lediglich die Schnittstellen, die auch beim Verbindungsaufbau durch die Anwendung bedient werden sollten, um beispielsweise die Ressourcenverwaltung aufrecht zu erhalten.

Um Aussagen über den Ort und die notwendige Form der Wissensrepräsentation zu präzisieren und um zu klären, welches Wissen für den Verbindungsaufbau und den Datentransport notwendig ist, sollen einige Beispielprozeduren angeführt werden. Zunächst wird der Verbindungsaufbau näher betrachtet:

Im folgenden wird der Anrufende mit Client bezeichnet, der angerufene als Server und mit Adresse bzw. Client-/Server-Adresse die Rechneradresse inkl. Prozeßadresse; ob es eine Transportrichtung gibt ist irrelevant. Der Verbindungsabbau wird nicht gesondert betrachtet, da er für die Analyse nicht von Bedeutung ist.

- Wird als Transportprotokoll TCP eingesetzt, muß der Server vor dem Verbindungsaufbau seine Bereitschaft signalisieren. Dies geschieht durch Binden eines Listen-Socket. Anschließend kann ein Client über das Connect-Kommando den Verbindungswunsch äußern, den der Server über das Accept-Kommando annehmen kann. Der Server muß also vor dem Verbindungsaufbau den Verbindungswunsch kennen; entweder ist der Server jederzeit bereit, eine Verbindung anzunehmen, oder es ist über einen weiteren Kommunikationsweg der Verbindungswunsch geäußert worden. In beiden Fällen muß dem Client die Serveradresse bekannt sein.
- Wird als Transportprotokoll RTP eingesetzt, findet kein echter Verbindungsaufbau statt, auch wenn in vielen Implementierungen der Client das Connect-Kommando aufruft, was er nur tut, um Daten eines bestimmten Servers herauszufiltern bzw. zu einem bestimmten Server zu senden und etwaige ICMP-Messages auszuwerten; das darunterliegende Protokoll (UDP[2]) ist verbindungslos. Für einen Datentransport muß der Server die Client-Adresse bzw. der Client die Serveradresse kennen. Üblicher-

[2] Ein verbindungsloses Transportprotokoll, das eine Schnittstelle zum Internet Protocol (IP) für den Benutzer bietet.

weise wird eine Prozeßadresse ausgehandelt, auf der Server und Client empfangen. Ein großer Vorteil des verbindungslosen Datentransportes sind die möglichen Mehrpunktbeziehungen. Ein Client muß dann nur die Mehrpunkt-Empfangs-Adresse kennen. Die hierarchische Struktur innerhalb einer Mehrpunktkommunikation entscheidet dann allein über die Existenz ausgezeichneter Server.

- Wird als Netzwerktechnologie ATM eingesetzt und kommt die Transportverbindung über die ATM-Signalisierung zustande, ist die Prozedur des Verbindungsaufbaus der bei einer TCP-Verbindung sehr ähnlich. Das Ergebnis ist eine SVC (Switched Virtual Circuit), die über eine VCI bzw. VPI gekennzeichnet wird. Auch hier muß nur der Client die Serveradresse kennen. Wird die ATM-Signalisierung nicht genutzt, kann ein Datentransport auf einer PVC (Permanent Virtual Circuit) erfolgen. Server bzw. Client müssen dann nur die VCI bzw. VPI der PVC aus ihrer Sicht erfahren. Auch über ATM sind Mehrpunktbeziehungen möglich, doch sind hier hierarchische Strukturen schon durch die Signalisierungsversion festgelegt. So ist bei der Signalisierung ATM UNI 3.1 die Aufnahme in eine Mehrpunktbeziehung nur durch einen ausgezeichneten Knoten möglich, hingegen kann bei ATM UNI 4.0 jeder Teilnehmer sich selbst in eine Mehrpunktbeziehung einbringen.

Um alle möglichen Varianten eines Verbindungsaufbaus in Abhängigkeit der eingesetzten Netzwerktechnologien abzudecken, muß Wissen explizit prozedural in Form von Regeln bzw. implizit prozedural in Form von Algorithmen vorliegen. Das Wissen über den Verbindungsaufbau ist unabhängig vom Wissen über die Methoden der eigentliche Datenübertragung, es kann somit autark vorliegen. Für jedes Protokoll könnte es spezielle Transportobjekte geben, die vom Transportdienst verwaltet werden. Sie können dann auch das notwendige deklarative Wissen enthalten, wie beispielsweise die Adressen der Verbindungsendpunkte in ihren speziellen Formaten.

Stream-Engine-Objekte

Nun wird der eigentliche Transport der Daten genauer betrachtet:

Die Verwaltung von Ressourcen und die Steuerung des Datenflusses bleibt zunächst unberücksichtigt.

- Sollen die Daten eines Stream-Objektes, das in Form einer Datei vorliegt, übertragen und beim Empfänger dargestellt werden, muß der Sender zunächst die entsprechende Datei öffnen, anschließend durch Leseoperationen den Dateiinhalt lesen und durch Sendeoperationen übertragen. Der Empfänger muß abhängig von der Darstellungsform das Ausgabegerät initialisieren, durch Empfangsoperationen die übertragenen Daten entgegennehmen und durch Ausgabeoperationen an das Ausgabegerät weiterleiteten.

- Sollen die Daten eines Stream-Objektes, das in Form eines Video-Encoders mit Kamera vorliegt, übertragen und auf der Empfängerseite aufgezeichnet werden, muß der Sender den Videoencoder initialisieren, und die vom Encoder gelieferten Daten durch Schreiboperationen übertragen. Der Empfänger muß zunächst eine Datei öffnen, dann durch Empfangsoperationen die übertragenen Daten entgegennehmen und durch Schreiboperationen in die Datei schreiben.

Abbildung 5.8 veranschaulicht die Datenpfade beider Beispiele noch einmal im Zustand nach der Initialisierung. Den „Motor“ der Übertragung stellen die Prozesse Lesen/Senden und Empfangen/Ausgeben dar. Das Problem, das sich hier offenkundig ergibt, ist das notwendige Spezialwissen z.B. über eine vom Media-Type abhängige Synchronisation zwischen Sender und Empfänger oder ob die Schnittstellen Quelle → Netz bzw. Netz → Senke auf Hardwareebene oder Softwareebene realisiert sind und wie sie angesprochen werden. Außerdem müssen solche Prozesse sehr effizient arbeiten, um mit den Systemressourcen wirtschaftlich zu haushalten. Diese Anforderungen werden ausschließlich durch die implizit prozedurale Wissensform erfüllt. Das Wissen liegt dann in Form von optimierten Algorithmen vor.

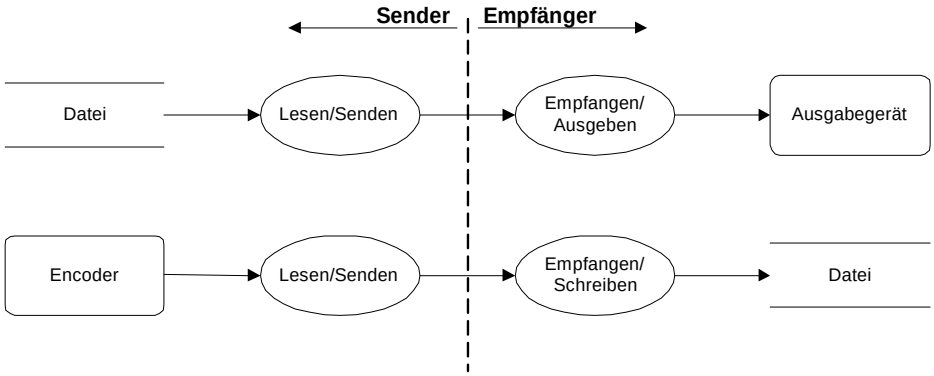

Abbildung 5.8 Beispiele für Datenpfade einer Datenübertragung

Die Betrachtungen zeigen auch sehr deutlich, daß aufgrund des Spezialwissens der eigentliche Antrieb für die Übertragung eines Stream-Objektes am besten vom Stream-Objekt selbst geliefert werden sollte. Damit wäre ein Stream-Objekt in der Lage, seinen Inhalt bzw. die dazu notwendigen Algorithmen selbst zu einem bestimmten Ort zu transportieren. Dies ist in dieser Form zur Zeit nicht mit vertretbaren Aufwand realisierbar. Ein Schritt in diese Richtung wäre jedoch, daß Wissen über die Übertragungsmethoden in speziellen Klassen zu kapseln und unabhängig vom restlichen System zur Verfügung zu Stellen, so daß Instanzen dieser Klassen von einem Transportobjekt für die Übertragung genutzt werden können, und dabei das Wissen leicht austauschbar bzw. erweiterbar ist. Dazu wird eine neue Objektklasse eingeführt, die Klasse *Stream-Engine*. Sie beherbergt den Antrieb für die Datenübertragung, wobei die Namengebung nicht zu dem Gedanken verleiten soll, der Datenpfad führe immer durch eine Stream-Engine-Instanz. Die Instanz könnte z.B. lediglich die Steuerung einer Treiber-Software kapseln, durch die der Datenpfad führt bzw. die wiederum einen Datentransfer via Hardware steuert. Abbildung 5.9 zeigt zwei Beispiele für die Führung des Datenpfades.

Die Stream-Engine soll eine Schnittstelle bieten, die Methoden zur Verbindung mit Stream-Objekten und Übertragungskanälen bereitstellt und die die Steuerung der Datenübertragung ermöglicht. Auch hier sollte, wie bei den Transportobjekten, die Spezialform einer passiven Stream-Engine eingeführt werden, falls die Anwendung den Datentransport durchführen soll.

Der Transportdienst verwaltet somit Transportobjekte und Stream-Engine-Objekte. Eine aktive Transportklasse enthält das a priori Wissen über den Aufbau einer Transportverbindung, eine aktive Stream-Objekt-Klasse verfügt über das Spezialwissen für den Datentransport. Damit ist eine Transportklasse in der Lage, Komponentenfähigkeiten zu realisieren, so daß der Transportdienst die Menge KF der Komponentenfähigkeiten an den Konnexionsdienst liefern kann, die dieser benötigt, um Beziehungen zu verträglichen Komponenten aufzubauen.

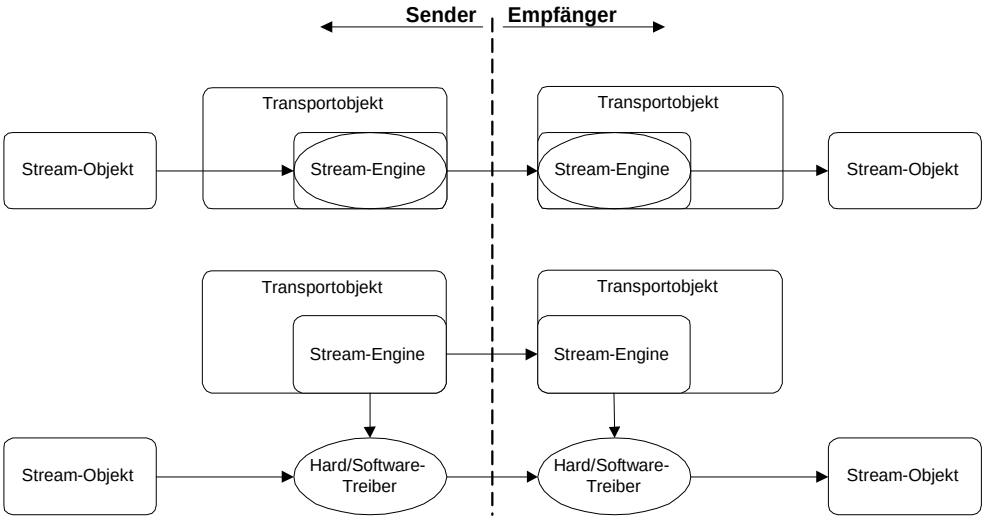

Abbildung 5.9 Beispiel für die Führung des Übertragungsdatenpfades

Die Existenz von Transportklassen und Stream-Engine-Klassen bestimmt die Menge der Fähigkeiten $KF(X) = \{KF_1(X), KF_2(X), \ldots, KF_k(X)\}$ einer DMSM-Systemkomponente. Für eine Ableitung der Komponentenfähigkeiten liefert die i-te Transportklasse die Beschreibungsdaten für das Protokoll $PR_i(X)$ und die Verbindungsmethode $VP_i(X)$. Die i-te Stream-Engine-Klasse liefert die Beschreibungsdaten der Datenquelle bzw. -senke $MT_i(X)$, $RI_i(X)$ und die Verbindungsmethode $VS_i(X)$.

Wird in der Menge $PR(X \mid MT_i(X))$ jedes $PR_j(X)$ aufgenommen, für das gilt

$VS_i(X) = VP_j(X) \; \forall \; j$

ergibt sich für die i-te Komponentenfähigkeit

$KF_i(X) = (MT_i(X), PR(X \mid MT_i(X)), RI_i(X))$

wobei

$PR_i(X)$ Das Übertragungsprotokoll, realisiert durch die i-te Transportklasse der Komponente X.

$VP_i(X)$ Die Verbindungsmethode der i-ten Transportklasse einer Komponente X.

$MT_i(X)$ *Media Type* der i-ten Stream-Engine-Klasse der Komponente X.

$RI_i(X)$ Datenrichtung der Fähigkeit i aus Sicht der Komponente X.

$VS_i(X)$ Die Verbindungsmethode der i-ten Stream-Engine-Klasse einer Komponente X.

Mit Verbindungsmethode wird die Datenpfadschnittstelle zwischen einer Stream-Engine und einem Transportobjekt bezeichnet. Ist die Datenpfadschnittstelle z.B. ein UDP-Verbindungsendpunkt, muß eine Stream-Engine diesen für einen Datentransport bedienen können, was z.B. bei einer RTP[3]/AVP[4]-Stream-Engine der Fall wäre.

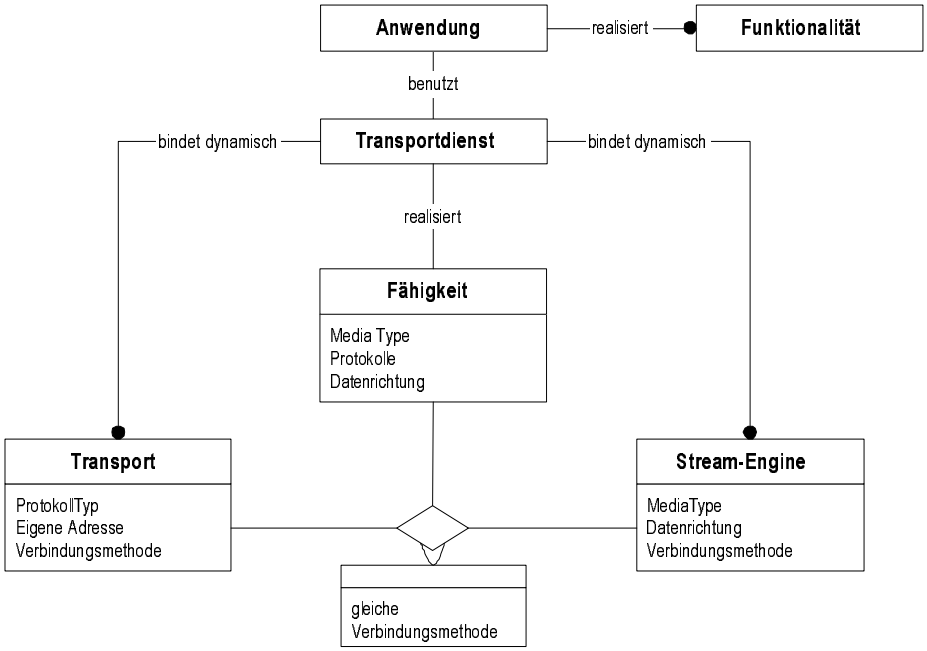

Abbildung 5.10 Objektmodell des Transportdienstes in OMT-Notation

[3] Das Realzeit-Transportprotokoll RTP ("Real-Time Transport Protocol") [RFC1889] bietet einen "Ende-zu-Ende"-Transportdienst für Daten mit Echtzeit-Anforderungen an.

[4] Audio- /Video-Profile

5.2.5 Steuerungsdienst

Der Steuerungsdienst verwaltet sogenannte Steuerungsobjekte und führt die für die Steuerung relevante Kommunikation mit der Applikation durch. Szenarien einer Steuerung innerhalb des DMSM-Systems können sehr unterschiedlicher Ausprägung sein, so ist für die Steuerung verteilter DM-Streams eine Mehrpunktbeziehung zwischen den Steuerobjekten notwendig, wie sie in Abbildung 5.11 dargestellt ist.

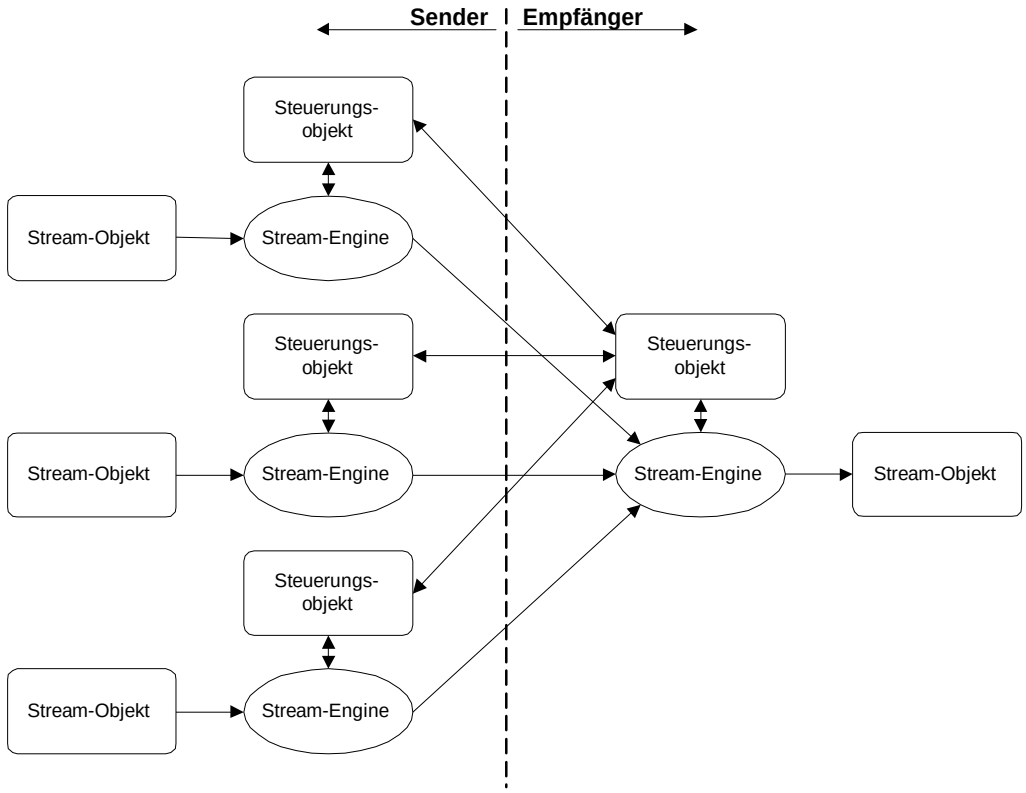

Abbildung 5.11 Beispiel für die Steuerung eines verteilten Streams

Steuerungsobjekte

Grundsätzlich beeinflussen Steuerungsobjekte Prozesse innerhalb der DMSMS-Komponenten; z.B. nehmen sie Einfluß auf die Datenübertragung eines Stream-Objektes. Auch eine ferngesteuerte Kamera kann sich innerhalb der Systemgrenzen einer DMSMS-Komponente befinden und über Steuerungsobjekte bedient werden.

Abbildung 5.11 läßt eine starke Symmetrie zwischen den einzelnen Steuerungsobjekten vermuten. Ein Steuerkommando wie START oder STOP muß jede steuernde Instanz ausführen, um den Datentransport zu starten bzw. zu stoppen. Eine solche Symmetrie würde eine spätere Realisierung erheblich vereinfachen.

Was in Abbildung 5.11 nicht dargestellt ist, sind die Aktivitäten, die Steuerereignisse auslösen und etwaige Reaktionen, die das Resultat einer Aktivität sein können. An dieser Stelle scheint die Symmetrie durchbrochen zu sein. In den meisten Fällen nimmt

immer eine ausgezeichnete steuernde Instanz Einfluß auf die Prozeßabläufe, nämlich diejenige, mit der ein Benutzer interagiert. Für alle anderen Steuerinstanzen besteht keine direkte Verbindung zu den Aktivitäten eines Benutzers.

Es muß zunächst geklärt werden, ob die Semantik einer direkten bzw. indirekten Aktivität für die Steuerung von Bedeutung ist. Für eine Untersuchung wird das folgende Szenario skizziert:

Es soll der Datenstrom einer Videokamera aufgezeichnet werden. Die Einspeisung der Videodaten erfolgt über einen Encoder, der als Stream-Objekt vom System A angeboten wird. Die Aufzeichnung der Daten wird auf System B durchgeführt, das ein Stream-Objekt anbietet, das explizit mit einer Datei assoziiert wird. Es wird vorausgesetzt, daß die Transportverbindung bereits besteht.

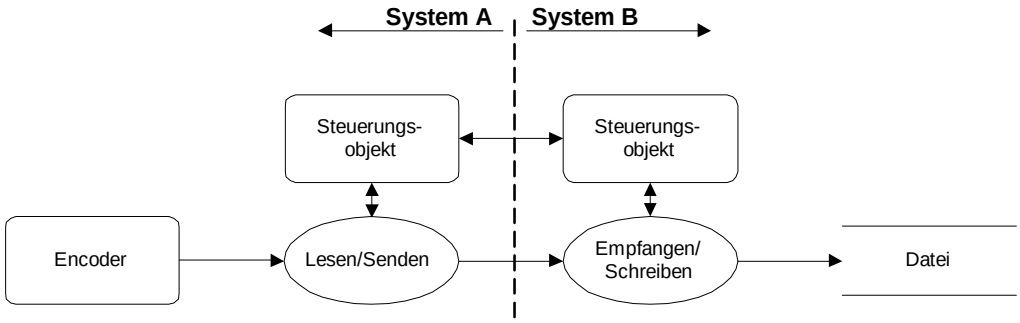

Abbildung 5.12 Beispielszenario zur Feststellung von Asymmetrien

Eine Aufzeichnung erfolgt erst dann, wenn die Prozesse *Lesen/Senden* und *Empfangen/Schreiben* gestartet wurden. Beide Prozesse müssen somit ein Start-Kommando erhalten. Dabei ist es gleichgültig, wer oder was das Start-Kommando auslöst. Die Ursache für eine Aktivität spielt also keine Rolle und stört die Symmetrie somit nicht, auch wenn das Szenario auf verteilte Stream-Objekte mit weitaus komplexeren Steuerkommandos ausgeweitet wird.

Bei Ereignissen verhält es sich ähnlich. Tritt z.B. der Ausnahmezustand *Aufzeichnungskapazität erschöpft* ein, müssen dies ebenfalls beide Prozesse erfahren. Ein Unterschied ergibt sich nur bei der Frage, wer dieses Ereignis auslösen kann. Dieser Unterschied kann z.B. dann von Bedeutung sein, wenn man das Szenario auf verteilte Stream-Objekte ausweitet. Wie soll sich das System verhalten, wenn bei nur einer Steuerinstanz der Ausnahmezustand *Aufzeichnungskapazität erschöpft* eintritt? Es könnte beispielsweise gefordert werden, daß die Aufzeichnung aller anderen Instanzen fortgeführt und für die ausgefallene eine Ersatzinstanz integriert wird. Eine derart differenzierte Behandlung durchbricht die Symmetrie in jedem Fall. Die Komplexität des Systems nimmt damit drastisch zu.

Einschränkung der Freiheitsgrade

Ein Weg, einer zunehmend kritischen Komplexität Einhalt zu gebieten, ist die Einschränkung von Freiheitsgraden. Ein augenfälliger Freiheitsgrad ist, daß aus Symmetriegründen die Ursache einer Aktivität bzw. eines Ereignisses keine Bedeutung haben soll. Im geschilderten Szenario hieße dieser Freiheitsgrad, daß es keine Rolle spielt, wer der Initiator der Aufzeichnung ist, er hat immer dieselbe Sicht auf die Steuerobjekte und dieselben Rechte. Es kann also keine Aussage über die tatsächliche Topologie der Steuerobjekte geäußert werden. Gerade dieser Aspekt ist eine formulierte Anforderung und sollte weitestgehend unangetastet bleiben, sofern es sinnvoll ist; das zu beurteilen bedarf einer neuen Fragestellung.

Wie lauten die tatsächlichen Beweggründe, eine Datenübertragung zwischen Stream-Objekten zu initiieren? Mögliche Antworten wären:

- Ich habe eine Datenquelle und suche ein Aufzeichnungsobjekt für den Mitschnitt.
- Ich suche ein Stream-Objekt speziellen Inhalts und möchte es lokal ausgeben.
- Ich möchte ein extern bereitgestellten Stream mitschneiden.
- Ich möchte einen Stream auf einem externen System ausgeben.
- usw.

Zwei Dinge sind aus den formulierten Motivationen erkennbar. Es gibt bei jedem denkbaren Beweggrund immer zwei Stream-Objekte und einen Initiator, wobei in den meisten Fällen der Initiator auch gleichzeitig der Besitzer mindestens eines der beiden Stream-Objekte ist. Weiterhin wird immer der Initiator für Aktivitäten innerhalb der beteiligten Systemkomponenten verantwortlich sein.

Eine Einschränkung der Freiheitsgrade kann erzielt werden, wenn man fordert: Der Initiator einer Datenübertragung muß auch gleichzeitig der alleinige Besitzer eines der an der Übertragung beteiligten Stream-Objekte sein. Wird als alleiniger Besitzer eines Stream-Objektes nur eine Komponente zugelassen, hätte dies zur Folge, daß eine Komponente, die nur im Besitz eines Teil-Streams ist, nie eine Datenübertragung mit einem externen Stream-Objekt und diesem Teil-Stream initiieren kann. Denn eine Komponente kann nicht allein Besitzer eines verteilten Stream-Objektes sein.

Diese Einschränkung ist vertretbar, wenn man sich vergegenwärtigt, in welchen Fällen ein Stream-Objekt verteilt vorliegen kann. Es kann nur im Falle von konservierten Daten verteilt vorliegen, so daß im Grunde genommen nur Aufzeichnungs- bzw. Wiedergabeobjekte von dieser Einschränkung betroffen sind.

Zusammenfassend kann gesagt werden: Der Freiheitsgrad, daß es keine Rolle spielt, wer der Initiator einer Datenübertragung und damit steuernde Instanz ist, gilt uneingeschränkt nur für konzentrierte Stream-Objekte. Für verteilte Stream-Objekte gilt: Eine Datenübertragung kann nur eingeleitet werden, wenn eines der daran beteiligten Stream-Objekte konzentriert vorliegt und der Initiator auch der Besitzer des Objektes ist. Es kann somit nie zu einer Datenübertragung zwischen verteilten Stream-Objekten kommen. In diesem Zusammenhang darf ein verteiltes Stream-Objekt nicht mit verteilten

Stream-Objekten, die keine Einheit bilden, verwechselt werden, so daß Multipunkt-Verbindungen von den vorgenommen Einschränkungen unberührt bleiben.

Aktivitäten- und Ereignisbus

Durch die vorgenommene Einschränkung der Freiheitsgrade wird erreicht, daß die Symmetrien ausgenutzt werden können, was die Berücksichtigung möglicher Topologien der Steuerungsobjekte unnötig macht. Dadurch gewinnt man die Freiheit, eine für den Zweck geeignete Topologie zu wählen. Geeignet ist z.B. eine Busstruktur. Alle Steuerungsobjekte schließen sich an einen Aktivitätenbus und an einen Ereignisbus an. Aktivitäten wirken sich in Form von Steuerkommandos bei allen angeschlossenen Instanzen auf die Zustände der an der Datenübertragung beteiligten Prozesse aus. Ereignisse können jeder Zeit auf dem Ereignisbus signalisiert werden, so daß jede Instanz darauf reagieren kann.

Der Aktivitätenbus darf im Gegensatz zum Ereignisbus nicht konsequent bidirektional ausgelegt werden. Steuerinstanzen, die sich nicht im Besitz des Benutzers befinden, können sich untereinander keine Steuersignale senden, sondern nur der Steuerinstanz des Benutzers. Diese Einschränkung resultiert aus den genannten Einschränkungen der Freiheitsgrade. Auf dem Ereignisbus steht es jeder Steuerinstanz frei, auf Ereignisse andere Instanzen zu reagieren.

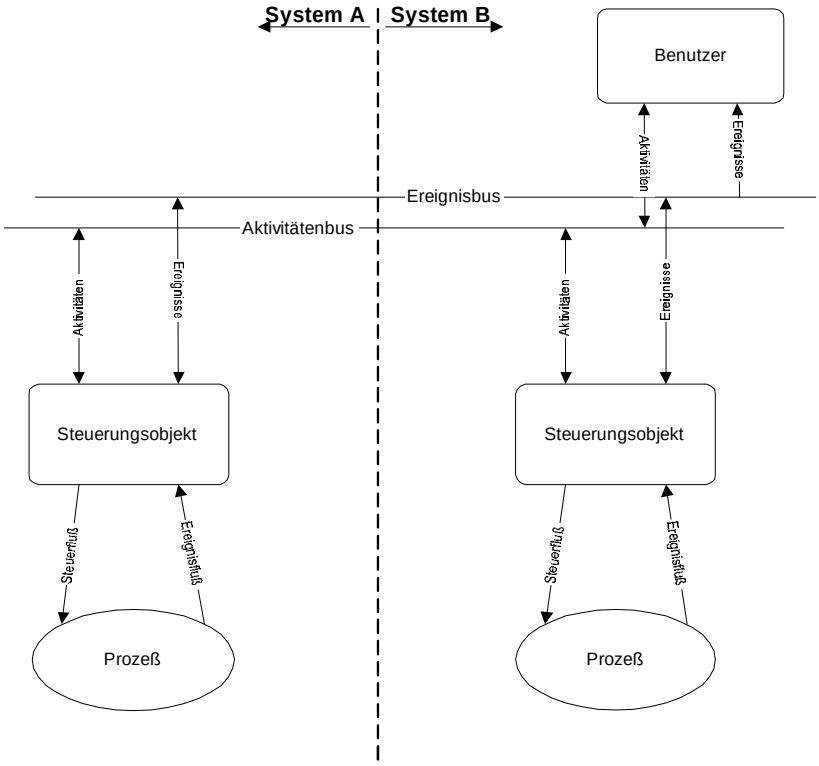

Abbildung 5.13 Organisation der Steuerobjekte in einer Busstruktur

Die Lage des Benutzers ist nun nicht mehr relevant, er kann die Steuersignale an jeder Stelle des Steuerbusses einspeisen und empfangen. Wenn die Benutzerinstanz ihren Ursprung in genau derselben Klasse hat, wie alle anderen Steuerungsobjekte und der Prozeß durch die Benutzerschnittstelle realisiert wird, ist die Symmetrie perfekt und die Implementierung fällt sehr einfach aus.

Einzelne Steuerbussysteme lassen sich über Steuerinstanzen koppeln. Damit können Baumstrukturen realisiert werden. Ein Anwendungsbeispiel wäre der Konverter (siehe auch Abbildung 5.14), der eine transparente Sicht in einen anderen Konnexionslayer bietet.

Die Steuerkommandos werden dabei immer von der Wurzel zu den Blättern übertragen und von den Blättern zu der Wurzel, nie aber von einem Blatt zu einem anderen. Die Konsequenz, die sich für das Beispiel des Konverters ergibt, ist, daß er nie von sich aus die Dateübertragung im Konnexionslayer 2 initiieren kann, obwohl er im Besitz eines konzentrierten Stream-Objektes ist.

Daß derartige Baumstrukturen möglich sind, sollte nicht dazu verleiten, sie ungehemmt zu nutzen, da die Signallaufzeiten mit jedem neuen Knoten steigen.

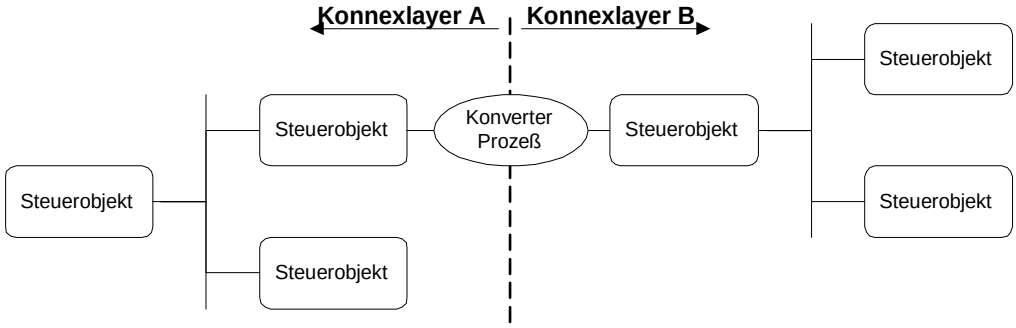

Abbildung 5.14 Baumstruktur bei Kopplung mehrerer Steuerbussysteme

Per Definition können signalisierte Ereignisse, egal von welcher Steuerinstanz sie geäußert werden, von jeder Steuerinstanz empfangen werden. Da dies nicht immer notwendig ist und die Entscheidungen über etwaige Reaktionen meist nur bei der Steuerinstanz des Benutzers sinnvoll getroffen werden können, ist ein Verschmelzen von Ereignisbus und Aktivitätenbus möglich. Eine vereinfachte Implementierung wäre der Nutzen dieser Maßnahme.

Eine Darstellung des Steuerungsdienstes in OMT-Notation wird an dieser Stelle nicht vorgenommen, da der Informationswert zu gering ist. Es stellt sich überhaupt die Frage, ob die bloße Verwaltung von Steuerungsobjekten, ohne daß jemand einen Nutzen davon hat, sinnvoll ist. Die Steuerobjekte kommunizieren eigenständig mit den Applikationen und sind eng an die zu steuernden Prozesse, die Stream-Engines, gekoppelt. Eine Kommunikation untereinander findet ausschließlich über den Steuerbus statt, so daß die Existenz einer übergeordneten Instanz nicht sinnvoll erscheint, solange ausschließlich Steuerobjekte der bisher besprochenen Kategorie eingesetzt werden.

5.2.6 Ressourcendienst

Mit Hilfe des Ressourcendienstes können Anfragen bzgl. verfügbarer Ressourcen gestellt werden. Dabei stehen einerseits die Zustände der einzelnen an einer Übertragung beteiligten Hardwarekomponenten zur Verfügung und andererseits alle von dem Transportdienst gestellten Ressourcenanforderungen. Der Ressourcendienst führt über sämtliche Anforderungen Statistiken, die von anderen Diensten und der Applikation genutzt werden können.

Ressourcen des DMSM-System

Was sind Ressourcen im Sinne des DMSM-Systems? Eine Ressource im Sinne des DMSM-Systems kann jede Hardwarekomponente sein, die an einer Datenübertragung beteiligt ist und ein begrenztes Leistungsvermögen bzw. eine begrenzte Kapazität besitzt. Auf einer abstrakteren Ebene sind demnach auch Stream-Objekte Ressourcen, da sie immer durch reale Ressourcen realisiert werden.

Die für eine Datenübertragung notwendigen oder die von einem Stream-Objekt genutzten Ressourcen sind beispielsweise Speicherkapazität, Festplattenkapazität, Prozessorleistung, Kanalkapazität, Anzahl der Audioencoder/-decoder, Anzahl der Videoencoder/-decoder usw.

Eines der verfolgten Ziele der Ressourcenverwaltung ist, eine Auskunft darüber geben zu können, ob eine Datenübertragung zustande kommen kann, und wenn ja, ob sie störungsfrei sein wird oder nicht. Mögliche Antworten der Ressourcenverwaltung auf eine Anfrage wären demnach:

- Die Übertragung kommt zustande, sie wird störungsfrei durchgeführt werden können.
- Die Übertragung kommt zustande, es gibt keine Garantie für die störungsfreie Durchführung.
- Die Übertragung kommt nicht zustande.

Der Nutzen der zweiten Antwort, wenn sie denn präzisiert bzgl. der Ursachen für die Störungen geäußert würde, läge in der Möglichkeit darauf zu reagieren, z.B. die Ressourcenanforderungen zu drosseln. Die zweite Antwort macht durch ihre unscharfe Formulierung auch die Schwierigkeit deutlich, präzise Auskunft über die Qualität einer Übertragung geben zu können. Dies liegt daran, daß das System unvorhersehbar belastet werden kann und Ressourcen, die zuvor zur Verfügung standen, nicht mehr verfügbar sind. Im Gegensatz dazu sind Ressourcen wie z.B. ein Audiokanal einfach zu handhaben, er ist entweder verfügbar oder nicht. Dieser Unterschied muß in einer abstrakten Beschreibung der Ressourcen Berücksichtigung finden.

Ressourcevariablen

Die abstrakte Beschreibung von Ressourcen kann z.B. durch sogenannte Ressourcevariablen erfolgen, von denen es zwei Kategorien gibt:

- diskrete Ressourcevariablen: $\{R: 0 < R < N \text{ mit } R(t), N \in \mathbb{N}\}$
- normierte kontinuierliche Ressourcevariablen: $\{r: 0 < r < 1 \text{ mit } r \in \mathbb{R}\}$

Der Wert einer Ressourcevariablen ist ein Maß für die verfügbare Kapazität einer Ressource zum gegenwärtigen Zeitpunkt. Jede Ressourcevariable muß einzeln abgefragt werden können. Außerdem muß ein Bewertungsmaß für die Gesamtauslastung definiert werden, in das alle Ressourcevariablen eingerechnet werden, um eine Auswahl geeigneter Komponenten aus einer Anzahl in Frage kommender Komponenten einfach zu gestalten.

Damit bei einem Ressourcenmangel eine Komponente noch genügend Handlungsspielraum besitzt, muß jede Ressourcevariable über einen Schwellwert verfügen, bei dem Maßnahmen zur Entlastung noch möglich sind, wie z.B. weitere Komponenten mit einem stark frequentierten Stream-Objekten zu versorgen.

Ressourcefunktionen

Ein großes Problem stellen die sequentiell verteilten Stream-Objekte dar. Da keine Prognose darüber abgegeben werden kann, wann die benötigten Ressourcen zur Ausspielung eines Teil-Streams tatsächlich beansprucht werden, sollte zumindest die Möglichkeit bestehen, Ressourcen zu buchen. Das bedeutet, daß es nicht mehr reicht, Ressourcen durch Variablen zum gegenwärtigen Zeitpunktes zu beschreiben, sondern daß Ressourcen durch Schätzfunktionen beschrieben werden müssen. Wie bei den Ressourcevariablen gibt es auch hier die Unterteilung in zwei Kategorien:

- diskrete Ressourcefunktionen: $\{R(t): 0 < R(t) < N \text{ mit } R(t), N \in \mathbb{N} \text{ und } t \geq 0\}$
- normierte kontinuierliche Ressourcefunktionen: $\{r(t): 0 < r(t) < 1 \text{ mit } r \in \mathbb{R} \text{ und } t \geq 0\}$

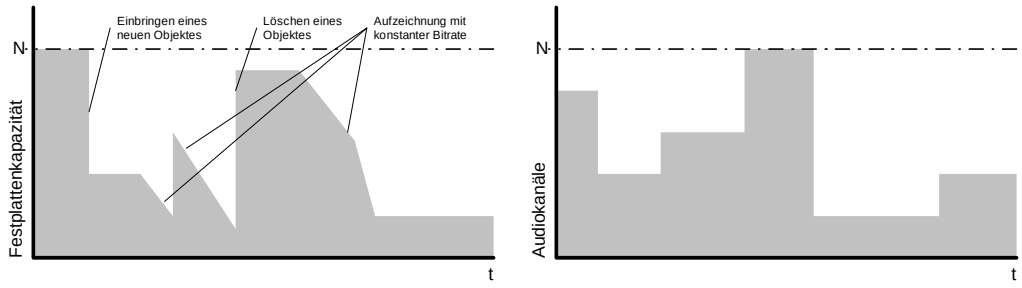

Abbildung 5.15 Beispiele für die Überlagerung diskreter Ressourcenfunktionen

Bei dem Versuch, derartige Funktionen zu bestimmen, wird man mit ähnlichen Schwierigkeiten konfrontiert wie bei der Wettervorhersage. Die Bestimmung solcher Funktionen sollte deshalb auf einfachen Modellen basieren und nur für die Ressourcen angewendet werden, die unter der Kontrolle der DMSMS-Komponente stehen. Ein Beispiel für eine schätzbare einfache diskrete Ressourcefunktion ist z.B. die der Festplattenkapazität. Wird beispielsweise eine Aufzeichnung zweier Live-Videoquellen mit den konstanten Bitraten CBR_1 und CBR_2 durchgeführt, könnten die Ressourcefunktionen lauten:

$R_1(t) = -tCBR_1 + R_1(0)$ *und* $R_2(t) = -tCBR_2 + R_2(0)$

Die Ressource Festplattenkapazität wird dann durch Überlagerung der für sie verfügbaren Funktionen innerhalb des interessierenden Intervalls abgeschätzt. Eine weitere anfragende Komponente könnte anhand der negativen Steigung erkennen, daß die Festplattenkapazität stetig sinkt und eine Entscheidung für die Beanspruchung der Komponente gegenüber alternativen Komponenten zurückstellen.

Ressourcen außerhalb des DMSMS-Wirkungsbereiches

Eine Ressourcefunktion, die für eine störungsfreie Übertragung sehr wichtig ist, aber nicht unter der direkten Kontrolle des DMSM-Systems steht und somit nicht bestimmt werden kann, ist die verfügbare Kanalkapazität einer Netzwerkverbindung. Die meisten Transportprotokolle wie z.B. TCP und UDP bieten nur ein einfaches Dienstmodell an, bei dem die Daten nach dem Best-Effort-Prinzip geliefert werden. Beim Best-Effort-Prinzip bestehen keine Garantien, ob bzw. wann das Versenden von Daten stattfindet. Dieses einfache Verfahren ist für das DMSM-System eher ungeeignet. Für Multimedia-Applikationen werden Dienste benötigt, die das zeitliche Übertragungsverhalten von multimedialen Daten über ein Netzwerk vorhersagbar machen. Die Dienste müssen somit unter anderem die Reservierung von Netzwerk-Ressourcen ermöglichen bzw. das kontrollierte Einbringen von Daten in ein Netzwerk erlauben.

Eine Ausnahme bildet ATM: Dort kann die Qualität eines Dienstes (QoS: Quality of Service) zugesichert werden. Vor dem eigentlichen Verbindungsaufbaus werden die Anforderungen an eine Verbindung in einer Flow-Spec-Datenstruktur [RFC1363] beschrieben und dem Serviceanbieter übermittelt. Erhält man dann den Zuschlag, ist einem die gewünschte Qualität sicher. Wünschenswert wäre ein ähnliches Vorgehen auch bei anderen Transportprotokollen wie beispielsweise TCP und UDP. Hier zeichnet sich ab, daß sich das RSVP (Resource ReSerVation Protocol) [SPEC95] [RSVP93] etabliert.

Die bestehende Transportschicht wird durch die Integration von RSVP ergänzt. Somit können zur Übertragung von Daten weiterhin das TCP- oder UDP-Protokoll verwendet werden. RSVP sorgt für den korrekten Ablauf bei der Reservierung von Netzwerk-Ressourcen. Die Anforderungen an eine Verbindung werden wie bei ATM über eine Flow-Spec-Datenstruktur beschrieben. Während einer Verbindungsanforderung können optional weitere für RSVP spezifische Informationen beigefügt werden, die eine Überreservierung verhindern sollen und außerdem festlegen, wie eine Reservierungsanforde-

rung *verschmolzen*[5] werden kann. Darüber hinaus können über Paketfilter Bandbreiten-Reservierungen auf bestimmte Sender begrenzt werden. D. h. nur die in einer Liste aufgeführten Sender dürfen Daten über den reservierten Pfad liefern.

Ob eine Netztechnologie die Möglichkeiten einer Ressourcenreservierung bietet oder nicht – eine Ressourcefunktion, die den aktuellen bzw. den zukünftigen Zustand des Netzwerkes abschätzt, kann nicht bestimmt werden. Es muß immer ein Reservierungsversuch durchgeführt werden, bevor man zu der Erkenntnis gelangt: Eine Datenübertragung ist möglich oder nicht.

Im Grunde führt eine solche Unbekannte die Aufgabe des Ressourcendienstes ad absurdum, denn eine echte Ressourcenreservierung ist damit nach wie vor nicht möglich. Es könnte dann bei allen Ressourcen ebenfalls nach dem Best-Effort-Prinzip vorgegangen werden. Entweder eine Komponente erhält den Audiokanal oder nicht. Bekommt sie ihn nicht, kann sie entsprechend darauf reagieren. Ein solches Vorgehen ist jedoch im höchsten Maße ineffizient, gerade im Hinblick darauf, daß die Suche nach alternativen Möglichkeiten nach Strategien verlangt und nicht durch bloßes Probieren zu bewältigen ist.

Aufgaben des Ressourcendienstes

Die Hauptaufgabe des Ressourcendienstes wird also in erster Linie nicht sein, das Zustandekommen von Verbindungen vorauszusagen, sondern abzuschätzen, welches die beste Variante aus einer Vielzahl von Alternativen ist.

Wo können sich überhaupt Alternativen ergeben? Grundsätzlich kann sich eine Alternative nur durch die Existenz eines Stream-Objekt-Repliks ergeben. Hierbei meint ein Replik auch ein Objekt gleichen Inhalts in einem anderen Format. Bedingt durch die Definition der Verträglichkeit von Komponenten (siehe dazu Kapitel 5.2.3) können sich dadurch Alternativen bei den Konnexionslayern und den Transportprotokollen ergeben.

Ein Auswahlkriterium sollte lauten, minimale Kosten bei maximaler Qualität zu erreichen. Diese Forderungen sind in sich paradox und stellen durch die schwierige Bewertung der Qualität kein Optimierungskriterium dar. Als bewertbare Größen eignen sich am besten die Kosten, zumal sie stark mit den benötigten Ressourcen korrelieren. Die Qualität sollte immer dem Anspruch genügen, und der wird meist maximal sein. Nach diesem Gesichtspunkt müßte immer die maximal mögliche Qualität angesetzt werden.

Ein weiteres Auswahlkriterium sollte Prioritäten höherer Ordnung berücksichtigen. Dazu ein Beispiel: Komponente A verfügt über zwei Stream-Objekte I und II, Komponente B verfügt über zwei Repliken der Stream-Objekte I und II von Komponente A und ein weiteres Stream-Objekt III. Die Komponente C wählt eines der Objekte I und II. Fällt wegen gleichmäßiger Auslastung der Komponente A und B die Entscheidung für Komponente B, kann es passieren, daß Stream-Objekt III für den Zugriff anderer blockiert wird, wenn während der Datenübertragung die Ressourcen für einen weiteren Zugriff nicht mehr reichen.

[5] [SPEC95] und [RSVP93] gehen weiter auf das *Verschmelzen* von Reservierungsanforderungen ein.

Dieser Fall mutet sehr konstruiert an und wird nur sehr selten auftreten. Außerdem kann leicht ein Konstrukt erzeugt werden, das eine richtige Entscheidung unmöglich macht. In dem genannten Beispiel müßte lediglich auf Komponente A ein weiteres Stream-Objekt IV untergebracht werden. Es reicht deshalb aus, das System mit der geringsten Auslastung zu wählen. Bei zufällig gleicher Auslastung zweier Komponenten sollte der Zufall entscheiden.

Damit der Ressourcendienst als Schiedsstelle effektiv wirken kann, muß er mit Spezialwissen ausgestattet werden. Das wären die Ressourcefunktionen, das Wissen über die von einem Stream-Objekt beanspruchten Hardwarekomponenten und ein Bewertungsmaß für die Qualität. Die besten Quellen für die Ressourcefunktionen und die Qualitätsmaße sind die Stream-Objekte selbst oder die mit ihnen assoziierten Stream-Engines. Für die Protokollwahl dient einerseits die bereits im Stream-Objekt integrierte Flow-Spec, andererseits könnte hier ebenfalls ein Bewertungsmaß definiert werden, um eine Vorauswahl treffen zu können.

Damit der Ressourcendienst als Datenbank für Hilfsdienste wie dem Replikationsdienst und dem Archivdienst wirken kann, muß er über sämtliche Stream-Objekte Zugriffsstatistiken führen.

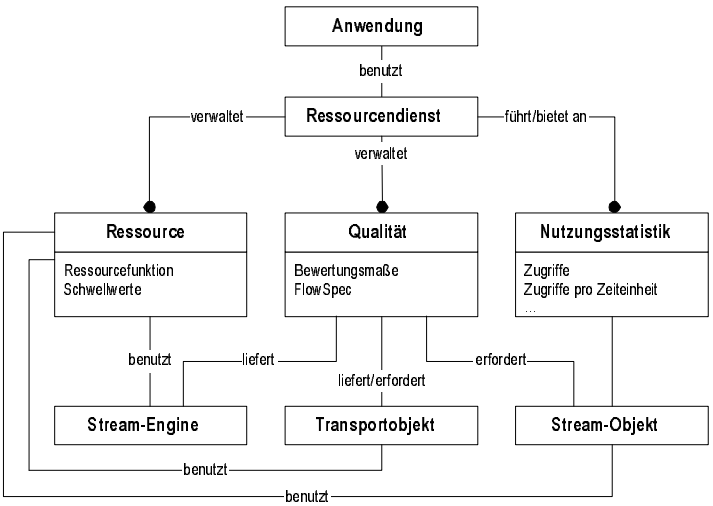

Abbildung 5.16 Objektmodell des Transportdienstes in OMT-Notation

Zusammenfassend kann gesagt werden: Der Registraturdienst hat in erster Linie die Aufgabe, bei mehrdeutigen Angeboten von Stream-Objekten dasjenige auszuwählen, das die maximale Qualität bietet und dabei geringste Kosten verursacht und dessen beteiligte Komponenten geringst möglich ausgelastet sind. Das dazu notwendige Spezialwissen wie Qualitätsmaße und Ressourcefunktionen bezieht der Ressourcendienst von den Stream-Objekten, den Transportobjekten und den Stream-Engine-Objekten. Die Anzahl der beteiligten Komponenten muß aus den Angeboten ersichtlich werden. Da die Ressourcefunktionen auch für den Zeitbereich $t > 0$ definiert sind, ermöglichen sie sequentiell verteilten Stream-Objekten ihre benötigten Ressourcen im voraus zu buchen. Eine Sonderrolle spielen die Netzressourcen, da der Zustand zu keinem Zeitpunkt von

einer Komponente in Erfahrung gebracht werden kann. Dieser Unsicherheitsfaktor muß zwangsläufig hingenommen werden.

Die zweite Aufgabe des Ressourcendienstes ist das Führen von Zugriffsstatistiken der von einer Komponente angebotenen Stream-Objekte. Die anfallenden Daten werden anderen Diensten bzw. den Applikationen zur Verfügung gestellt.

5.2.7 Registraturdienst

Der Registraturdienst dient dem Erfassen neuer Stream- und Dienstobjekte. Applikationen können sich über diesen Dienst beim DMSM-System anmelden, die DMSM-Systemkomponente bzgl. ihrer Anforderungen konfigurieren und das Einbinden neuer Objekte erfahren bzw. veranlassen.

Im Gegensatz zu den anderen Diensten, die vornehmlich Funktionalität bieten, stellt der Registraturdienst hauptsächlich eine Datenbank dar. Der Inhalt der Datenbank wird mitbestimmt durch den Kontext der anderen Dienste. Die Daten lokaler Stream-Objekte werden aus Informationen des Transportdienstes ermittelt; sie stehen im Kontext der verfügbaren Transportobjekte und Stream-Engine-Objekte. Die Daten externer Stream-Objekte werden aus den Angeboten extrahiert, die der Konnexionsdienst entgegennimmt und stehen damit im Kontext der existenten Konnexionslayer und der gestellten Anfragen.

Das Registrieren der Applikationen bzw. die Konfiguration entsprechend den Bedürfnissen einer Applikation ist nichts weiter als das Einprägen bestimmter Attribute, die dann den funktionalen Ablauf des eigenen bzw. der anderen Dienste beeinflussen. Sie können quasi als Filterbedingungen angesehen werden. Sie nehmen zusätzlich Einfluß auf den Inhalt der Datenbank.

Das Aufgabengebiet des Registraturdienstes gliedert sich in zwei Teile: die Verwaltung lokaler Stream-Objekte und die Verwaltung externer Stream-Objekte. Die Zweiteilung resultiert aus der Diskussion über die Beschränkung der Freiheitsgrade zur Minderung der Komplexität in Kapitel 5.2.5, die zum Ergebnis hatte, daß bei einem Datentransport mindesten ein konzentriertes lokales Stream-Objekt beteiligt sein muß.

Die Verwaltung lokaler Stream-Objekte

Die Verwaltung lokaler Stream-Objekte sammelt über alle lokal verfügbaren Stream-Objekte beschreibende Informationen, stellt Assoziationen zwischen Stream-Objekten und deren Daten her und ordnet sie in der Datenbank ein. Die Gewinnung der beschreibenden Daten sowie der Assoziationen birgt jedoch einige Probleme in sich, deren Natur erst bei genauerer Betrachtung verschiedenartiger Stream-Objekte erkennbar wird. Dazu werden exemplarisch einige Stream-Objekte angeführt:

- Das Stream-Objekt konservierter Daten: Der Inhalt des Stream-Objektes wird durch eine Datei auf einem Datenträger repräsentiert. Die Methoden für den Datentransport implementiert ein Stream-Engine-Objekt. Die Assoziation von Stream-Engine und Datei bildet quasi erst das Stream-Objekt. Dies führt dazu, daß

mit einem Stream-Engine-Objekt mehrere Stream-Objekte in das System integriert werden können.

- Das Stream-Objekt einer Live-Datenquelle: Der Inhalt des Stream-Objektes wird durch die Live-Datenquelle repräsentiert. Die Datenquelle ist beispielsweise eine Hardwarekomponente. Die Methoden für den Datentransport werden auch hier von einem Stream-Engine-Objekt implementiert. Die Assoziation von Stream-Engine und Stream-Objekt ist implizit gegeben. Es wird somit durch Bereitstellen eines Stream-Engine-Objektes nur ein Stream-Objekt in das System integriert.
- Das Stream-Objekt für eine Datenaufzeichnung: Einen Inhalt dieses Stream-Objektes gibt es nicht. Die Methoden zur Datenaufzeichnung implementiert ebenfalls ein Stream-Engine-Objekt. Die Assoziation von Stream-Engine und Stream-Objekt ist implizit gegeben. Aus diesem Grund wird durch Einfügen eines Stream-Engine-Objektes zu Datenaufzeichnung ein Stream-Objekt in das System integriert.

Bei der Betrachtung des Stream-Objektes zur Aufzeichnung stellt sich die Frage: Ist ein Stream-Objekt ohne Inhalt überhaupt als solches einzuordnen? Es ist quasi ein Metazustand eines Stream-Objektes konservierter Daten, denn in dem Augenblick, in dem der Datenfluß während einer Aufzeichnung erstarrt, haben die Daten nichts mehr mit dem Aufzeichnungsobjekt zu tun, sie sind dann Inhalt eines Stream-Objektes konservierter Daten.

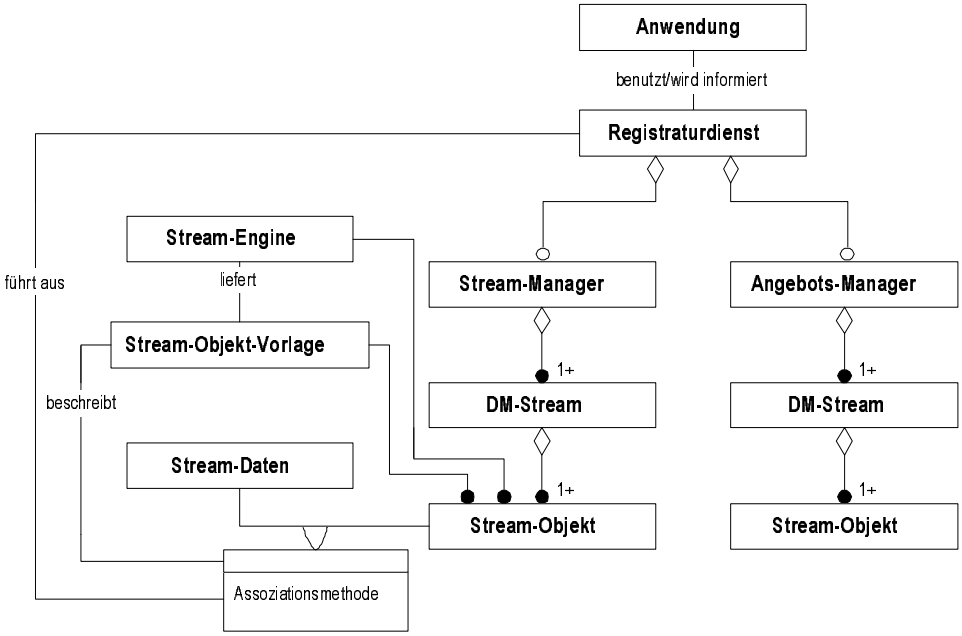

Abbildung 5.17 Objektmodell des Registraturdienstes in OMT-Notation

Um diesem Dilemma und dem Problem der multiplen Assoziation zwischen Stream-Engine-Objekt und Stream-Objekten konservierter Daten zu entrinnen, werden für eine abstraktere Beschreibung der Stream-Objekte die sogenannten Stream-Objektvorlagen eingeführt. Aufgabe der Stream-Objektvorlagen ist es, dem Registraturdienst alle für die Verwaltung von Stream-Objekten notwendigen Informationen zu geben. Eine Vorla-

ge ist eine Art Formular, das von einem Stream-Engine-Objekt vor seiner Integration in das System ausgefüllt werden muß. Anschließend liefert es während der Instanziierung eines Stream-Objektes alle notwendigen Daten, die es dem Registraturdienst ermöglichen z.B. Assoziationen zwischen Stream-Objekten und seinen Daten zu erkennen.

Eine Stream-Objektvorlage besitzt sämtliche Attribute einer Stream-Objektklasse und gibt ihnen eine allgemeinere Bedeutung. So gilt beispielsweise für die Flow-Spec einer Objektvorlage, daß alle mit der assoziierten Stream-Engine transportierten Daten, die dort beschriebenen Dienstgüte erfordern. Darüber hinaus sind Attribute notwendig, die Auskunft darüber geben, ob die Assoziationsmethode explizit oder implizit ist, ob die Anzahl der möglichen lokalen Instanzen begrenzt ist und ob Instanzen verteilt vorliegen können. Letzteres Attribut folgt aus der in Kaptitel 5.2.2 formulierten Definition, daß verteilte Stream-Objekte dieselbe UUID besitzen, so daß über dieses Attribut festgelegt werden kann, ob das System die UUID automatisch zuweist, damit jede Kopie eines Stream-Engine-Objektes eindeutig bleibt oder ob die UUID von der Objektvorlage geliefert wird.

Die Verwaltung externer Stream Objekte

Die Verwaltung externer Stream Objekte beschränkt sich auf die Annahme und Einordnung der empfangenen Angebote aller Konnexionslayer.

Die Einordnung, egal ob es sich um ein externes oder lokales Stream-Objekt handelt, muß unter Berücksichtigung der folgenden zwei Punkte geschehen:

- Die Existenz von Repliken ist möglich. Repliken sind vom Objektanbieter erstellte Kopien, die dieselbe UUID besitzen.
- Jedes Stream-Objekt kann ein Teil-Stream sein. Es muß dann im Kontext eines DM-Streams gesehen werden (siehe auch Abbildung 5.4).

5.3 Dynamische Modellierung

Während der Objektmodellierung wurden die statischen Strukturen des DMSM-Systems untersucht, das heißt, die Struktur seiner Objekte und ihrer Relationen zueinander zu einem bestimmten Zeitpunkt. Nun sollen die Veränderungen von Objekten und ihren Relationen untersucht werden, die sich im Laufe der Zeit ergeben. Es werden Konzepte entwickelt zur Regelung des Kontrollflusses, der Interaktionen und der Ausführungsreihenfolge von Operationen parallel zueinander aktiver Objekte innerhalb des DMSM-Systems.

5.3.1 Ablauf der Komponentenintegration

Eine der Aufgaben des Konnexionsdienstes ist es, zwischen DMSM-Systemkomponenten Beziehungen herzustellen. Er ist somit für die Ausprägung der Konnexionslayer verantwortlich.

Eine Beziehung kann zwischen zwei DMSMS-Komponenten zustande kommen, wenn der Medien-Typ der zu übertragenden Informationen derselbe ist und beide Komponenten mindestens ein identisches Protokoll über ein gemeinsames Netz beherrschen. So wurde es in Kapitel 5.2.3 durch die Definition der Verträglichkeit festgelegt. Dies setzt voraus, daß jede im DMSM-System befindliche Komponente durch Präsentation ihrer Fähigkeiten bekannt ist. Um die Anzahl der integrierbaren Komponenten im System nicht zu beschränken, darf keine Komponente Informationen über eine andere speichern. Dadurch ergibt sich ein Kommunikationsparadoxon: Es sollen Beziehungen eingegangen werden, aber keine Komponente darf etwas über eine andere wissen. Es kann gezeigt werden, daß mit Hilfe des Konzeptes der Konnexionslayer dieses Paradoxon gelöst wird. Zur Veranschaulichung der Lösung wird im folgenden eine Analogie mit Hilfe der Funktechnik[6] gebildet:

Es existiert eine unbekannte Zahl von Teilnehmern, die sich nicht kennen und eine endliche Anzahl verschiedener Interessen haben. Der Wille jedes Teilnehmers ist es, Kontakt zu anderen Teilnehmern mit den gleichen Interessen aufzunehmen, um darüber Informationen auszutauschen. Alles, was ein Teilnehmer für die Kommunikation besitzt, sind zwei Funkgeräte. Das eine besitzt eine feste Sende- und Empfangsfrequenz, die sogenannte Basisfrequenz, die bei allen anderen Teilnehmern identisch ist, und das andere besitzt eine variable Sende- und Empfangsfrequenz.

Würden alle Teilnehmer über die Basisfrequenz kommunizieren, wäre das Chaos perfekt, da nicht mehr als ein Funkgerät auf derselben Frequenz senden kann, sofern sie in Reichweite sind. Deshalb darf diese Frequenz nur so selten und so kurz wie möglich belegt werden. Um die Basisfrequenz zu entlasten, gruppieren sich zunächst die Teilnehmer gleicher Interessenlage, indem sie sich mit Hilfe des zweiten Funkgerätes auf einer Frequenz treffen, die von der Basisfrequenz verschieden ist. Wie kann sich nun ein neuer Teilnehmer, der noch keine Kenntnis über die Gruppenfrequenzen besitzt, in eine Gruppe integrieren?

Für den folgenden Algorithmus wird vorausgesetzt, daß jeder Teilnehmer einer Gruppe einen eindeutigen Rang besitzt. Der Rang ist eine Nummer zwischen 0 und der Anzahl der Gruppenteilnehmer. Desweiteren wird vorausgesetzt, daß jeder Gruppenteilnehmer den maximalen Rang kennt.

Für die Aufnahme eines Teilnehmers in eine Gruppe gleicher Interessenlage äußert der Teilnehmer zunächst auf der Basisfrequenz sein Interesse und nennt gleichzeitig eine reelle Zufallszahl zwischen 0 und 1, den sogenannten normierten Rang und eine Frequenz, auf der er Antworten mit Hilfe des zweiten Funkgerätes empfängt. Alle anderen Teilnehmer registrieren die Äußerung und überprüfen, ob eine Übereinstimmung mit der eigenen Interessenlage besteht. Wenn nein, wird nichts weiter unternommen. Wenn ja, wird der normierte Rang umgerechnet, indem er mit dem maximalen multipliziert wird.

[6] Die Analogie mit Hilfe der Funktechnik wurde bewußt gewählt, da das Multicasting [TANE96] in Computernetzen ähnliche Eigenschaften wie die von Funkkreisen aufweist. Die Problematik, daß mehrere Teilnehmer innerhalb der Reichweite auf einer Frequenz zeitgleich senden könnten, ist bei Computernetzen nicht so kritisch zu bewerten. Hier sind es eher Leistungsspitzen, die die Verarbeitung zeitgleicher Sendungen gefährden könnten. Der Algorithmus muß also nicht das zeitgleiche Senden, sondern die zur Senderzahl proportionalen Leistungsspitzen vermeiden.

Der maximale Rang wir anschließend um eins erhöht. Der Teilnehmer, bei dem der errechnete Rang mit dem eigenen übereinstimmt, meldet sich auf der vorgeschlagenen Frequenz zurück und teilt die Frequenz und den neuen maximalen Rang der Gruppe gleicher Interessenlage mit. Der neue Teilnehmer setzt seinen Rang auf den neuen maximalen Rang der Gruppe.

Damit innerhalb der Rangfolge keine Lücken entstehen, muß jeder Teilnehmer sich beim Verlassen seiner Gruppe abmelden. Dabei äußert er seinen Rang, so daß jeder verbleibende Teilnehmer, der einen Rang höher als den Geäußerten besitzt, ihn um eins vermindern kann, womit die Lücke gefüllt wäre. Darüber hinaus vermindert jeder Teilnehmer der Gruppe den gemerkten maximalen Rang um eins.

Ist ein Teilnehmer in eine Gruppe Gleichgesinnter integriert worden, kann er gezielt Fragen stellen. Während er eine Frage auf der Gruppenfrequenz äußert, gibt er gleichzeitig eine neue Frequenz an, auf der er eine Antwort erwartet. Teilnehmer, die auf die Anfrage antworten möchten, tun dies auf der angegebenen Frequenz, um die Gruppenfrequenz zu entlasten. Falls ein Teilnehmer spontan neue Erkenntnisse veröffentlichen will, weil er der Meinung ist, daß sie andere Gruppenteilnehmer interessieren könnte, kann er das ebenfalls auf der Gruppenfrequenz tun.

Der beschriebene Algorithmus funktioniert nur, solange ein ideales Funknetz existiert und die Gruppenteilnehmer sich diszipliniert abmelden. Da dies in der Realität nicht vorausgesetzt werden kann, können in der Rangfolge Lücken entstehen. Dies führt zu dem Risiko, daß während der Integration in eine Gruppe der zufällig gewählte Rang nicht besetzt ist und eine Antwort ausbleibt. Eine Lösung des Problems kann sein, daß alle Teilnehmer einer Gruppe bei der Integration eines neuen Teilnehmers einen Zeitgeber starten, nach dessen Ablauf eine Rückmeldung erfolgt. Die Zeitdauer wird in Abhängigkeit der Distanz des eigenen zu dem zufällig gewählten Rang gesetzt. Nachdem der neue Teilnehmer eine Antwort erhalten hat, signalisiert er auf der Gruppenfrequenz oder der Basisfrequenz das Ende des Integrationsvorganges, damit alle Teilnehmer, deren Zeitgeber noch läuft, diesen deaktivieren können. Ein Schließen von einmal entstanden Lücken lohnt sich nicht, da mit zunehmender Anzahl von Teilnehmern das Treffen einer Lücke immer unwahrscheinlicher wird und das Entstehen von Lücken eher die Ausnahme ist.

Erhält ein Teilnehmer keine Antwort auf der von ihm vorgeschlagenen Frequenz, nachdem er sein Interesse auf der Basisfrequenz kundgetan hat, muß er davon ausgehen, daß noch keine Gruppe gleicher Interessenlage existiert. Er bestimmt dann selbst die Gruppenfrequenz, wenn ein weiterer Teilnehmer die gleiche Interessenlage auf der Basisfrequenz propagiert. Er ruft somit eine neue Gruppe ins Leben. Die Möglichkeit, daß jeder Teilnehmer die Macht besitzt, eine Gruppe ins Leben zu rufen, birgt die Gefahr der Clusterbildung in sich, wenn beispielsweise während der Integration neuer Teilnehmer durch atmosphärische Störungen die Reichweite eingeschränkt war, so daß die neuen Teilnehmer eine weitere Gruppe auf einer Frequenz eröffnet haben, die von der Frequenz der bereits existierenden Gruppe verschieden ist. Die Clusterbildung läßt sich nicht vermeiden; es kann aber, wenn notwendig, die Möglichkeit geschaffen werden,

Cluster zusammenzuführen. Die Kriterien, die eine Zusammenführung zur Folge haben, müßten dann noch diskutiert werden.

Der eben dargestellte Algorithmus ermöglicht den Aufbau von Beziehungen, ohne daß jemals Informationen über einen anderen Teilnehmer gespeichert werden. Überträgt man den Algorithmus auf das DMSM-System, entsprechen den Teilnehmern die Komponenten, den Interessenlagen die Fähigkeiten und den Gruppen die Konnexionslayer.

Die graphische Darstellung des Algorithmus erfolgt als Zustandsdiagramm[7]:

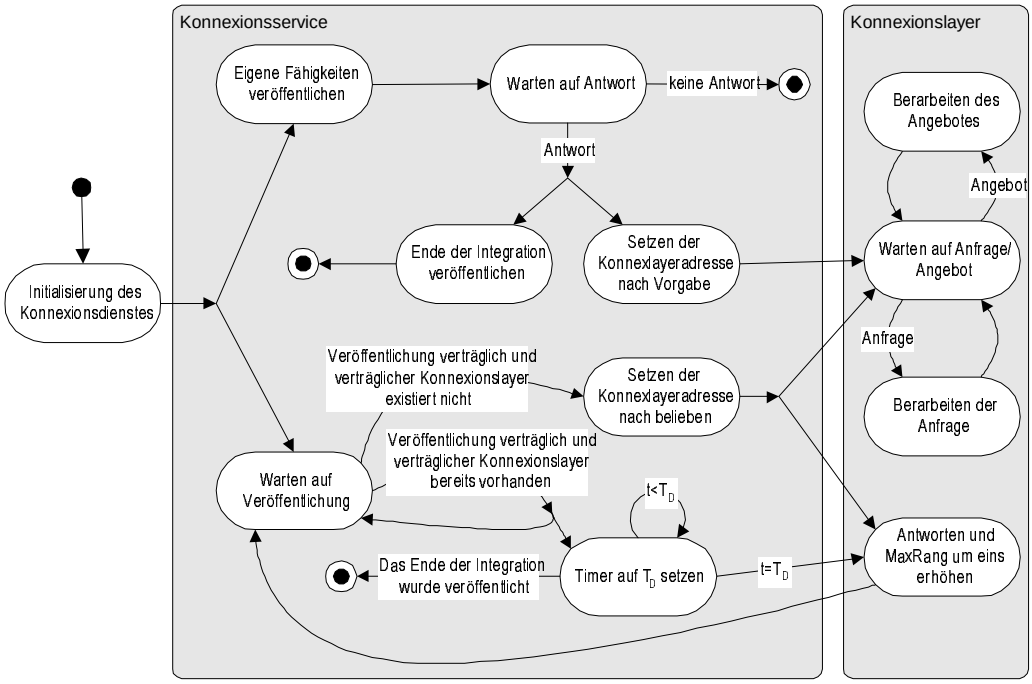

Abbildung 5.18 Zustandsdiagramm der Komponentenintegration

Die in Abbildung 5.18 benutzte Variable T_D ist die Zeit, um die eine Antwort verzögert wird. Sie sollte nach der folgenden Formel berechnet werden:

$$T_D = T_{2R}\left(2 \cdot \left|R - rR_{Max}\right| + f_A\left(R - rR_{Max}\right)\right)$$

wobei gilt:

$T_{2R} = const.$	zweifache mittlere Reaktionszeit
$r \in \Re$	normierter Rang
$R \in \aleph$	eigner Rang
$R_{Max} \in \aleph$	aktueller maximaler Rang

[7] Abweichend von der üblichen Notation werden parallele Vorgänge immer mit der Aufspaltung eines Pfeils eingeleitet.

$$f_A(x) = \begin{cases} -1 \; \textit{für } x < 0 \\ 0 \quad \textit{sonst} \end{cases}$$ Aktivierungsfunktion

Die zweifache mittlere Reaktionszeit T_{2R} muß empirisch ermittelt werden und sollte etwas höher gewählt werden als die tatsächlich ermittelte. Das Produkt aus reeller und ganzer Zahl rR_{Max} muß durch Streichen der Nachkommastellen wieder in eine Ganzzahl gewandelt werden.

5.3.2 Anfrage- und Angebotsbehandlung

Nachdem sich eine Komponente in das DMSM-System integriert hat, kann sie Anfragen stellen und Angebote entgegennehmen. Anfragen werden grundsätzlich an jede Komponente eines Konnexionslayers verteilt. Angebote hingegen werden nur an einen Anfragenden gesandt, sofern die Anfrage die Ursache des Angebotes war. Angebote werden nur dann an alle Teilnehmer gesandt, wenn eine Komponente ein Angebot bewußt veröffentlichen will.

Eine Anfrage bewirkt bei jeder Komponente des gleichen Konnexionslayers, daß nach Übereinstimmungen zwischen den in der Anfrage formulierten Anforderungen und den beschreibenden Metadaten der lokalen Stream-Objekte gesucht wird. Das Ergebnis können kein, ein oder mehrere Stream-Objekte sein. Wenn mindestens ein Stream-Objekt der Anfrage entspricht, wird es in Form eines Angebotes an den Anfragenden übermittelt. Dem Angebot wird eine Kanalkennzeichnung angefügt, über die ggf. der Datentransport initiiert werden kann.

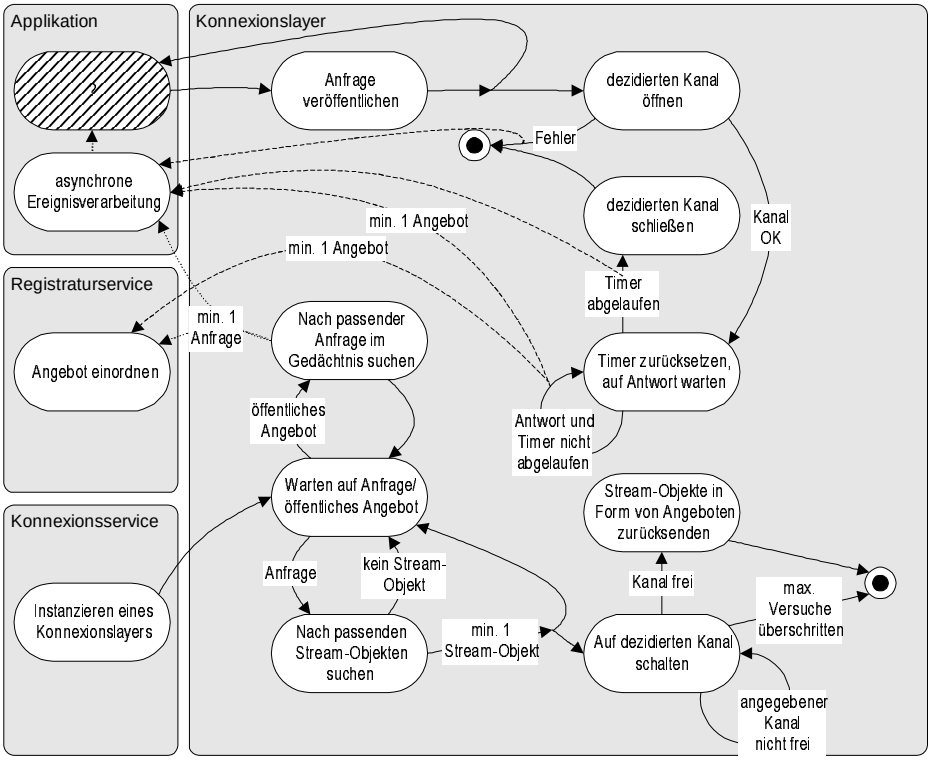

Abbildung 5.19 Zustandsdiagramm der Angebots- und Anfragebehandlung

Angebote, die gezielt an einen Anfragenden gesandt werden, übermittelt der Anbieter auf einem vom Fragenden vorgeschlagenen dezidierten Kanal. Der Kanal wird von der sendenden Komponente solange belegt, bis er alle Angebotsdaten übermittelt hat. Wenn der Kanal für andere Angebotsäußerungen wieder freigegeben wird, erfolgt eine asynchrone Benachrichtigung an die Applikation. Will eine Komponente Angebote auf dem dezidierten Kanal zeitgleich mit einer anderen äußern, so muß eine der beiden so lange warten, bis die Übermittlung der anderen abgeschlossen ist.

Wie lange der dezidierte Kanal geöffnet bleibt, sprich Angebote geäußert werden können, liegt im Ermessen des Empfängers. Ein mögliches Vorgehen wäre, bei jeder Inanspruchnahme des Rückmeldekanals einen Zeitgeber zu setzten, nach dessen Ablauf der Kanal geschlossen wird. Wenn man davon ausgeht, daß Anfragen von jeder Komponente so schnell wie möglich beantwortet werden, ist dieses Vorgehen durchaus praktikabel.

Da ein Kommunikationskanal, unabhängig von seiner Realisierung, in irgendeiner Weise gekennzeichnet ist, muß gewährleistet sein, daß die Kennzeichnung über einen Zeitraum hinweg eindeutig ist, so daß eine Fehlleitung eines Angebotes ausgeschlossen ist.

Damit veröffentlichte Angebote nutzbringend sind, muß der Konnexionsservice die Möglichkeit anbieten, Anfragen im Gedächtnis zu behalten. Er kann dann die Applikation asynchron benachrichtigen, falls ein veröffentlichtes Angebot einer Anfrage im Gedächtnis entspricht.

Die Verarbeitung der asynchronen Ereignisse[8], die bei Zustandsänderungen ggf. signalisiert werden, sollte von der Applikation nicht unbedingt implementiert werden müssen.

5.3.3 Initiierung eines Datentransportes

Sobald mindestens ein Angebot eingegangen ist, besteht die Möglichkeit, einen Datentransport zwischen einem DM-Stream-Objekt[9] des Stream-Managers und einem DM-Stream-Objekt des Angebotsmanagers einzuleiten. Die Vorgabe des lokalen DM-Stream-Objektes und des Angebotes ist Aufgabe der Applikation.

Eine Transportverbindung kann nur dann zustande kommen, wenn das lokale DM-Stream-Objekt ein konzentriertes Stream-Objekt enthält und der Initiator auch der Besitzer des Stream-Objektes ist (siehe auch Kapitel 0 Abschnitt *Einschränkung der Freiheitsgrade*). Damit sind aus Sicht des Initiators nur 1:M-Beziehungen möglich, aus Sicht der Gegenstellen nur 1:1-Beziehungen.

Einleiten eines Datentransportes aus Sicht des Initiators

Mit der Annahme, der DM-Stream des gewählten Angebotes enthält M Stream-Objekte, müssen zunächst M Kommunikationsverbindungen zu den Komponenten[10] aufgebaut werden, die im Besitz der M Stream-Objekte sind. Die Adresse für den Verbindungsaufbau wird den Angeboten entnommen. Desweiteren besteht die Möglichkeit, daß sich innerhalb der M Stream-Objekte Repliken anderer Stream-Objekte befinden. Über die Kommunikationsverbindungen müssen folgende Dinge geklärt werden:

- *Spezifikation der gewählten Objekte*: Dazu werden die Angebote an ihre Absender zurückgesandt. Zuvor muß jedoch die UUID neu gesetzt werden und ggf. die Kanalnummer und die Kanalanzahl bestimmt werden. Die neue UUID bzw. Kanalspezifikation kann dann von der Gegenstelle für die Erzeugung explizit assoziierter verteilter Stream-Objekte benutzt werden, z.B. bei einem verteilten Aufzeichnungsobjekt. Wahlweise können alle beschreibenden Daten spezifiziert werden, die ein später erzeugtes Objekt erbt, doch dies sollte vornehmlich der Applikation überlassen werden.
- *Transportrichtung*: Nach der Objektspezifikation muß in jeder Gegenstelle die mit dem Stream-Objekt assoziierte Stream-Engine ermittelt und deren Transportrichtung dem Initiator mitgeteilt werden. Die Transportrichtung muß vor dem Instanziieren der Transportobjekte bekannt sein, da in Abhängigkeit der Transportrichtung sich Unterschiede im Verbindungsaufbau ergeben können.
- *Transportprotokoll*: Über die Verbindungsmethode der mit dem Stream-Objekt assoziierten Stream-Engine können in jeder Gegenstelle alle in Frage kommenden Pro-

[8] In den Zustandsdiagrammen sind signalisierte Ereignisse durch gestrichelte Linien gekennzeichnet.

[9] siehe auch Kapitel 5.2.7 bzw. Abbildung 5.17

[10] Eine Komponente, die an einer Datenübertragung beteiligt ist, wird im folgenden auch mit Gegenstelle bezeichnet.

tokolle ermittelt werden. Die Wahl des geeigneten Protokolls wird beim Initiator durchgeführt, so daß alle Gegenstellen die Menge aller Protokolle dem Initiator übermitteln müssen. Den Protokollen wird ein normiertes Bewertungsmaß beigefügt, das der Ressourcendienst der jeweiligen Gegenstelle liefern muß. Es wird immer die Kombination von Stream-Engine und Protokoll in Abhängigkeit aller zur Zeit bekannten Ressourcefunktionen bewertet.

- *Auswahl geeigneter Repliken*: Hat der Initiator die Beschreibung aller Transportprotokolle mit ihren Bewertungsmaßen entgegengenommen, wählt er die Repliken mit der besten Bewertung aus. Zu allen Gegenstellen der nicht gewählten Stream-Objekte wird die Kommunikationsverbindung beendet. Allen anderen Gegenstellen wird die Wahl des Protokolls mitgeteilt.
- *Adressen der Übertragungsendpunkte*: Nachdem die Wahl des Protokolls jeder Gegenstelle bekannt gemacht wurde, werden zunächst nur die Transportobjekte der Gegenstellen instanziiert. Jedes Transportobjekt muß eine Methode bereitstellen, die Auskunft über die eigene Adresse gibt. Die über den Methodenaufruf ermittelte Adresse wird dann dem Initiator mitgeteilt. Dieser instanziiert nun seinerseits die Transportobjekte und gibt beim Verbindungsaufbau die von den Gegenstellen gelieferten Adressen an.

Wenn eine Kommunikationsverbindung zu mindestens einer Gegenstelle zustande kommt und alle aufgeführten Punkte erfolgreich mit den erreichbaren Gegenstellen geklärt werden konnten, wird das lokale Stream-Engine-Objekt instanziiert. Es importiert das lokale Stream-Objekt des Stream-Managers und die von den Transportobjekten exportierten Verbindungsendpunkte. Abschließend erhält die Applikation die Steuerungsschnittstelle des Steuerungsobjektes, das von dem Stream-Engine-Objekt exportiert wird.

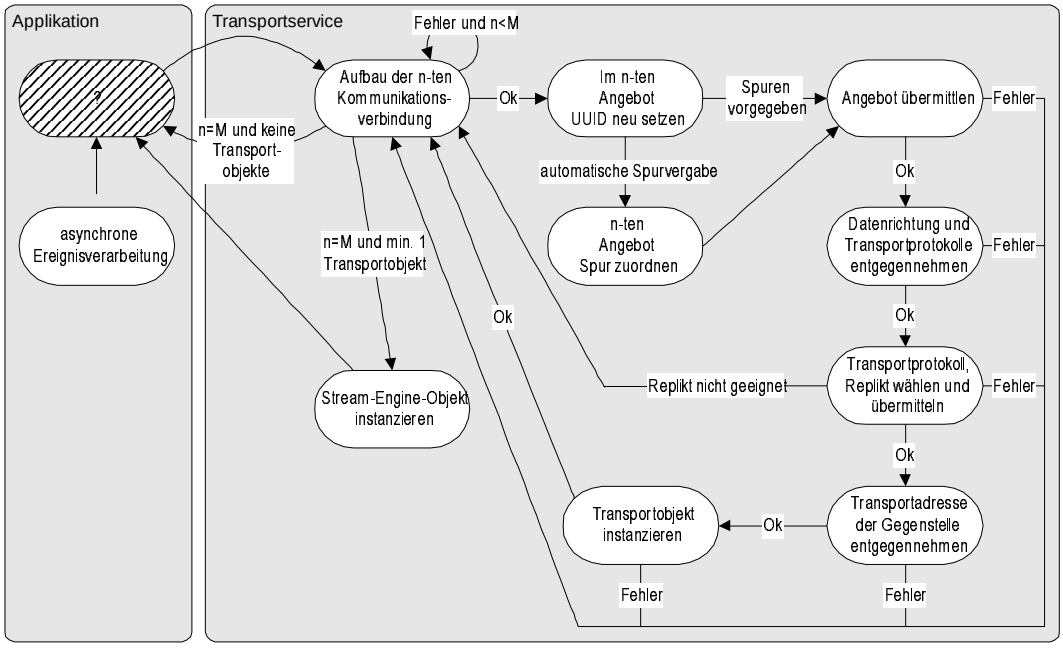

Abbildung 5.20 Einleiten eines Datentransportes aus Sicht des Initiators

Einleiten eines Datentransportes aus Sicht einer Gegenstelle

Wenn eine Komponente ein Angebot veröffentlicht, muß sie eine Adresse angeben, über die im Falle einer Datentransportanforderung eine Kommunikationsverbindung aufgebaut werden kann.

Mit Hilfe der Kommunikationsverbindung wird zunächst das gewünschte Stream-Objekt spezifiziert, indem es in Form des eigenen Angebotes entgegengenommen wird. Die mit dem angebotenen Stream-Objekt assoziierte Stream-Engine wird ermittelt und deren Datenrichtung zurückgesandt. Alle in Frage kommenden Protokolle werden in Kombination mit der Stream-Engine vom Ressourcendienst bewertet. Anschließend werden die Protokolle mit ihren Bewertungsmaßen dem Initiator mitgeteilt.

Bei allen Komponenten, die nach der Protokollwahl bzw. Bewertung für die Datenübertragung nicht in Frage kommen, wird die Kommunikationsverbindung an dieser Stelle beendet. Bei den übrigen Komponenten wird die Entscheidung für ein bestimmtes Protokoll entgegengenommen, das entsprechende Transportobjekt instanziiert, die eigene Adresse ermittelt und an den Initiator zurück gesandt.

Ist bisher kein Fehler aufgetreten, wird das mit dem Angebot assoziierte Stream-Engine-Objekt instanziiert und das Angebot selbst, sowie der vom Transportobjekt exportierte Verbindungsendpunkt übergeben.

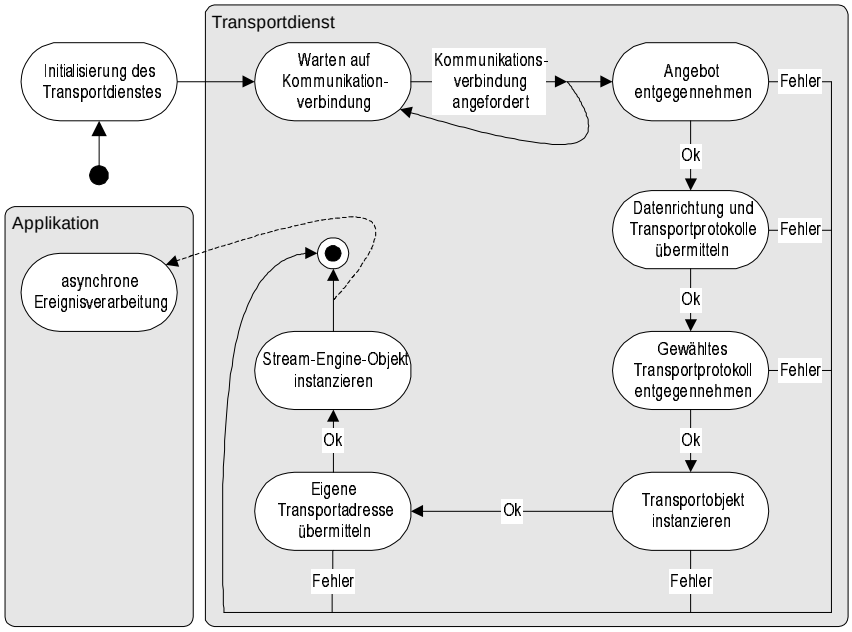

Abbildung 5.21 Einleiten eines Datenransportes aus Sicht einer Gegenstelle

5.4 Funktionale Modellierung

Das funktionale Modell soll alle wichtigen Datenflüsse und Datentransformationen innerhalb des DMSM-Systems aufzeigen.

Für die graphische Darstellung des funktionalen Modells werden Datenflußpläne in etwas modifizierter Form eingesetzt. Datenflüsse, die über Kommunikationsverbindungen stattfinden, werden explizit durch Trennung der Instanzen dargestellt. Dadurch kann es vorkommen, daß Prozesse mehrfach auftauchen. Mit Hilfe dieser Darstellung können diejenigen Objekte identifiziert werden, die später über eine Netzverbindung übertragen werden müssen. Derartige Objekte sollten während der Implementierung mit einer Methode ausgestattet werden, die das Objekt effizient in einen kompakten seriellen Datenstrom überführt.

5.4.1 Konnexionsdienst

Während der Initialisierungsphase der Applikation muß der Konnexionsdienst instanziiert werden. Aufgrund der informellen Abhängigkeit vom Transportdienst instanziiert der Konnexionsdienst zunächst den Transportdienst. Auf Basis der Analyse aller dynamischen Objekte liefert der Transportdienst alle Informationen, die für die Instanziierung der Konnexionslayer notwendig sind. Ergebnis der Instanziierung eines Konnexionslayers muß eine Rückmeldeadresse sein, über die, von einer anderen Komponente aus, eine Kommunikationsverbindung aufgebaut werden kann.

Die mit einem Konnexionslayer assoziierte Fähigkeit wird zusammen mit der Rückmeldeadresse des Layers veröffentlicht. Alle Komponenten, die bereits die Initialisierungsphase abgeschlossen haben, nehmen die Veröffentlichung entgegen. Eine von ihnen

antwortet über die Rückmeldeadresse und liefert die aktuelle Layeradresse. Falls diese noch nicht gesetzt ist, wird sie vor der Auslieferung initialisiert (für den genauen zeitlichen Ablauf siehe Abbildung 5.18).

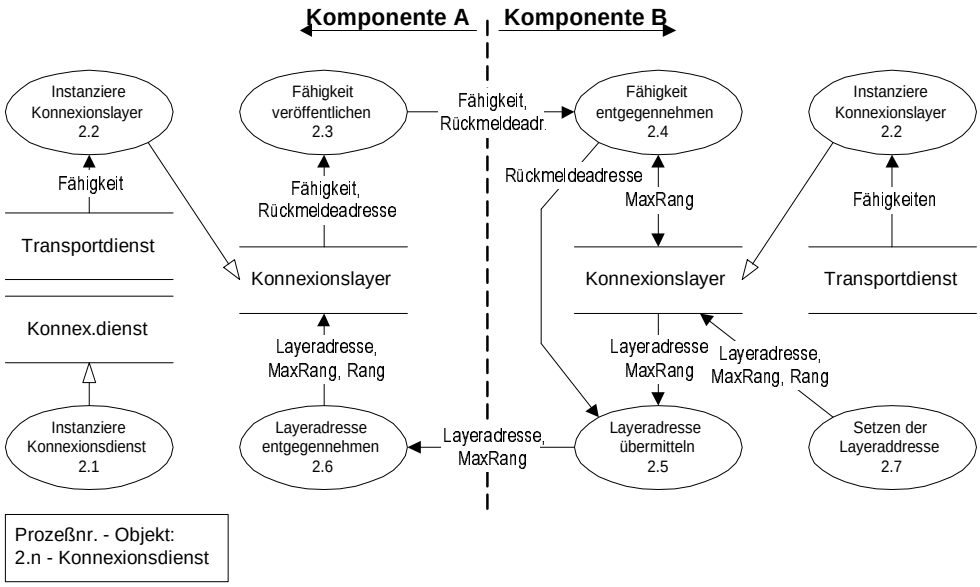

Abbildung 5.22 Datenflußplan des Konnexionsdienstes

Für die Veröffentlichungen ist eine allgemein bekannte Adresse notwendig, die sogenannte Basisadresse. Die Basisadresse muß während der Implementierung festgelegt werden und sollte danach nicht mehr verändert werden. Eine Ausnahme könnte sein, daß in einem Netz mehrere unabhängige DMSM-Systeme eingesetzt werden sollen. Dann sollte für jedes DMSM-System eine eigene Basisadresse gewählt werden.

Es ist darauf zu achten, daß der Datenfluß während der Veröffentlichung der Fähigkeiten durch die 1:M-Beziehung multipliziert wird. Die Rückmeldung ist immer eine konkurrenzlose 1:1-Beziehung und damit unkritisch.

5.4.2 Konnexionslayer

Ein Konnexionslayer befindet sich solange in der Initialisierungsphase, bis er eine Antwort auf die Veröffentlichung seiner Fähigkeit erhalten hat oder bis ein weiterer verträglicher Konnexionslayer seine Fähigkeit veröffentlicht hat. Erst dann können Anfragen veröffentlicht und Angebote entgegengenommen werden.

Eine Anfrage wird von der Applikation instanziiert. Bevor sie zur Veröffentlichung an den Konnexionsdienst übergeben wird, können Filterbedingungen formuliert werden. Wenn eine leere Anfrage übergeben wird, heißt dies, daß alle verfügbaren Informationen in allen zur anfragenden Komponente verträglichen Konnexionslayern im gesamten DMSM-System abgerufen werden sollen.

Eine Anfrage gelangt über den Konnexionsdienst zu den Konnexionslayern, die den Filterbedingungen entsprechen. Dort wird ein Prozeß abgesetzt, der die Antworten auf die Anfragen entgegennimmt. Dieser Prozeß legt auch die Rückmeldeadresse fest.

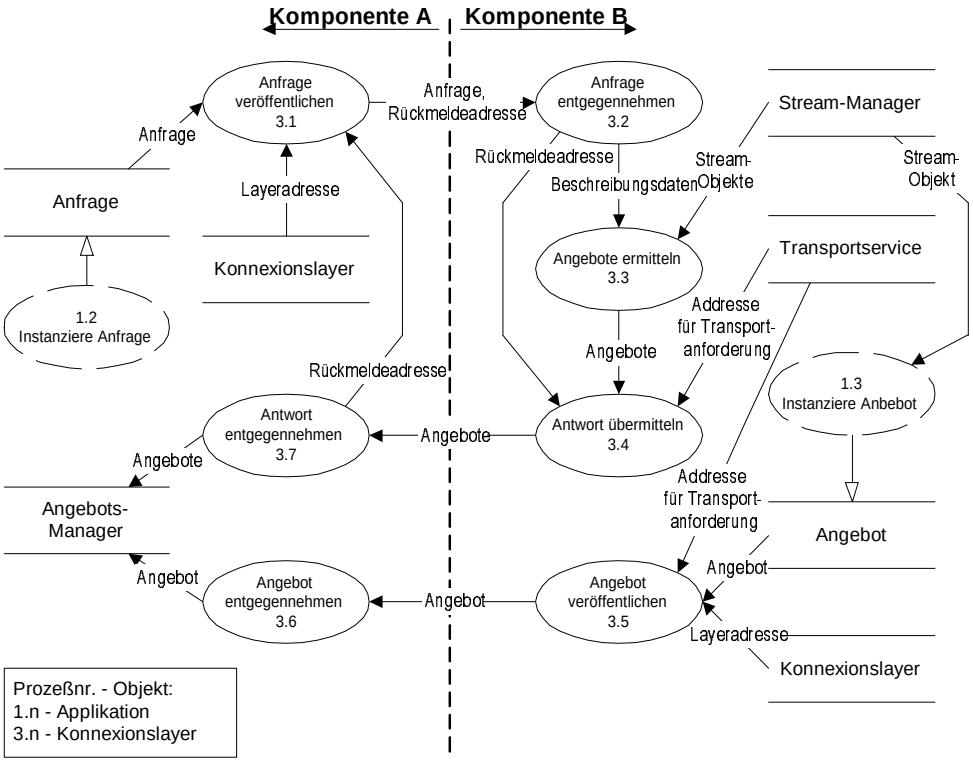

Abbildung 5.23 Datenflußplan eines Konnexionslayers

Die Anfrage wird zusammen mit der Rückmeldeadresse über die Layeradresse veröffentlicht. Dadurch gelangt die Anfrage zu allen verträglichen Komponenten. Diese nehmen die Anfrage entgegen und ermitteln alle der Anfrage entsprechenden Angebote. Jedem Angebot wird eine Adresse beigefügt, über die der Datentransport eingeleitet werden kann. Diese Adresse stellt der Transportdienst bereit. Nachdem alle in Frage kommenden Angebote ermittelt wurden, wird ein dezidierter Rückkanal zur Rückmeldeadresse geöffnet und die Angebote übertragen.

Auch hier ist darauf zu achten, daß der Datenfluß während der Veröffentlichung einer Anfrage durch die 1:M-Beziehung multipliziert wird. Kritisch ist die Rückmeldung zu bewerten. Eine Rückmeldung ist zwar eine 1:1-Beziehung, sie erfolgt jedoch in Konkurrenz mit anderen Komponenten. Es ist deshalb bei der Implementierung darauf zu achten, daß die dadurch resultierenden Leistungsspitzen entzerrt werden müssen. Zum einen kann der Prozeß 3.7 (siehe Abbildung 5.23) als Server mit paralleler Verarbeitung implementiert werden, zum anderen kann auf der Senderseite durch eine zufällige Streuung des Rückmeldezeitpunktes für Entlastung gesorgt werden.

Angebote können noch über einen zweiten Datenpfad verteilt werden. Dazu instanziiert die Applikation ein Angebot und veröffentlicht es über die Layeradresse. Dadurch erreicht das Angebot alle verträglichen Komponenten.

Empfangene Angebote werden, unabhängig von ihrer Quelle, dem Angebotsmanager übergeben. Die Applikation kann dann über den Registraturdienst auf die Angebote zugreifen.

5.4.3 Registraturdienst

Sobald Angebote entgegengenommen bzw. geäußert werden sollen, muß der Registraturdienst instanziiert sein. Mit dem Registraturdienst wird der Stream-Manager und der Angebotsmanager aktiviert. Der Angebotsmanager ordnet lediglich alle eingehenden Angebote ein, so daß die Applikation komfortabel darauf zugreifen kann.

Bevor eine Komponente selbst Angebote versenden bzw. veröffentlichen kann, müssen Stream-Objekte im Stream-Manager registriert worden sein. Hier ist zwischen implizit und explizit assoziierten Stream-Objekten zu unterscheiden (siehe hierzu auch Kapitel 5.2.7 Abschnitt: *Die Verwaltung lokaler Stream-Objekte*). Die Registratur von Stream-Objekten beginnt mit einer Anfrage an den Transportdienst. Von ihm werden sämtliche Stream-Engine-Objekte angefordert.

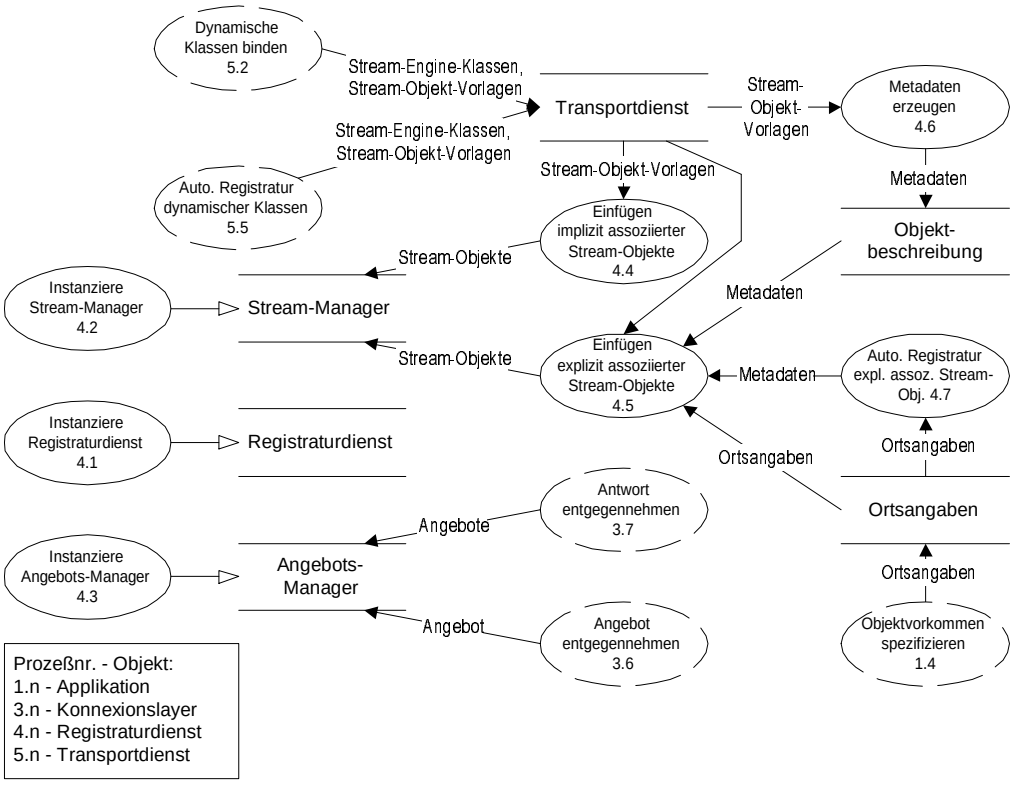

Abbildung 5.24 Datenflußplan des Registraturdienstes

Stream-Engine-Objekte, die ihre Daten implizit assoziieren, können in Form eines Stream-Objektes sofort dem Stream-Manager übergeben werden.

Bei Stream-Engine-Objekten, die ihre Daten explizit assoziieren, müssen zunächst die Verbindungen zwischen Daten und Objekt erstellt werden. Dazu muß die Applikation dem Registraturdienst Ortsangaben übermitteln. Durch die Ortsangaben wird lediglich festgelegt, wo sich Daten konkreter Stream-Objekte befinden können. Mit Hilfe der Hinweise, die ein Stream-Engine-Objekt über eine Stream-Objektvorlage äußert, werden die angegebenen Orte nach eventuellen Objektdaten durchsucht. Werden Objektdaten gefunden, wird versucht, Beschreibungsdaten zu extrahieren. Sind keine Beschreibungsdaten verfügbar, werden sie aus der Stream-Objektvorlage und den Objektdaten

abgeleitet. Auf Basis der Beschreibungsdaten erstellt der Registraturdienst dann das Stream-Objekt, das dem Stream-Manager übermittelt wird.

Sobald die Applikation eine Ortsangabe äußert, muß der Registraturdienst einen Überwachungsprozeß aktivieren. Sobald neue Daten an einem der angegebenen Orte auftauchen, versucht der Überwachungsprozeß, diese in Zusammenhang mit einem Stream-Engine-Objekt zu bringen. Gelingt dies, wird das daraus resultierende Stream-Objekt automatisch dem Stream-Manager übergeben.

Das automatische Einfügen implizit assoziierter Stream-Objekte sollte ebenfalls unterstützt werden. Registriert der Transportdienst ein neues dynamisches Objekte und handelt es sich um ein Stream-Engine-Objekt, wird der gesamte Integrationsprozeß genauso durchlaufen wie bei der Initialisierungsphase.

5.4.4 Transportdienst

Die Konfiguration des Transportdienstes entscheidet darüber, in welchem Konnexionslayer eine Komponente integriert werden kann. Die Konfiguration des Transportdienst ist abhängig von den Fähigkeiten einer Komponente; die Fähigkeiten wiederum sind ausschließlich von dynamisch gebundenen Objekten abhängig. Dynamisch gebundene Objekte sind Transportobjekte und Stream-Engine-Objekte. Sie liegen beispielsweise in Form von Objektdateien auf einem Datenträger. Im ersten Schritt werden lediglich Informationen über die Klassen aus den Objektdateien gewonnen. Falls es sich um ein Stream-Engine-Objekt handelt, wird darüber hinaus das Objekt beauftragt, eine Stream-Objektvorlage auszufüllen. Die Klasseninformationen und die Stream-Objektvorlagen stehen während der gesamten Laufzeit der Applikation und anderen Prozessen innerhalb einer DMSMS-Komponente zur Verfügung.

Soll nun ein Datentransport zwischen einem lokalen und einem externen Stream-Objekt eingeleitet werden, muß ein Angebot des Angebotsmanagers und ein Stream-Objekt des Stream-Managers angegeben werden. Die Auswahl der Objekte wird beim Initiator durch die Applikation vorgenommen, bei den Gegenstellen muß der Transportdienst auf Basis der Informationen des zurückgesandten Angebotes die Objekte selbst ermitteln.

Mit Hilfe der Stream-Objekte werden die assoziierten Stream-Engine-Objekte ermittelt; diese wiederum lassen durch die Verbindungsmethode die Auswahl der Transportobjekte zu. Die Entscheidung für oder wider ein Transportobjekt wird im Laufe des Prozesses 5.8 (siehe Abbildung 5.25) gefällt. Nachdem das Transportobjekt instanziiert wurde, liefert es bzw. erhält es die Adresse für einen Verbindungsaufbau. Die Stream-Engine-Objekte erhalten nach der Instanziierung das zurückgesandte Angebot bzw. das lokale Stream-Objekt und die Verbindungsendpunkte.

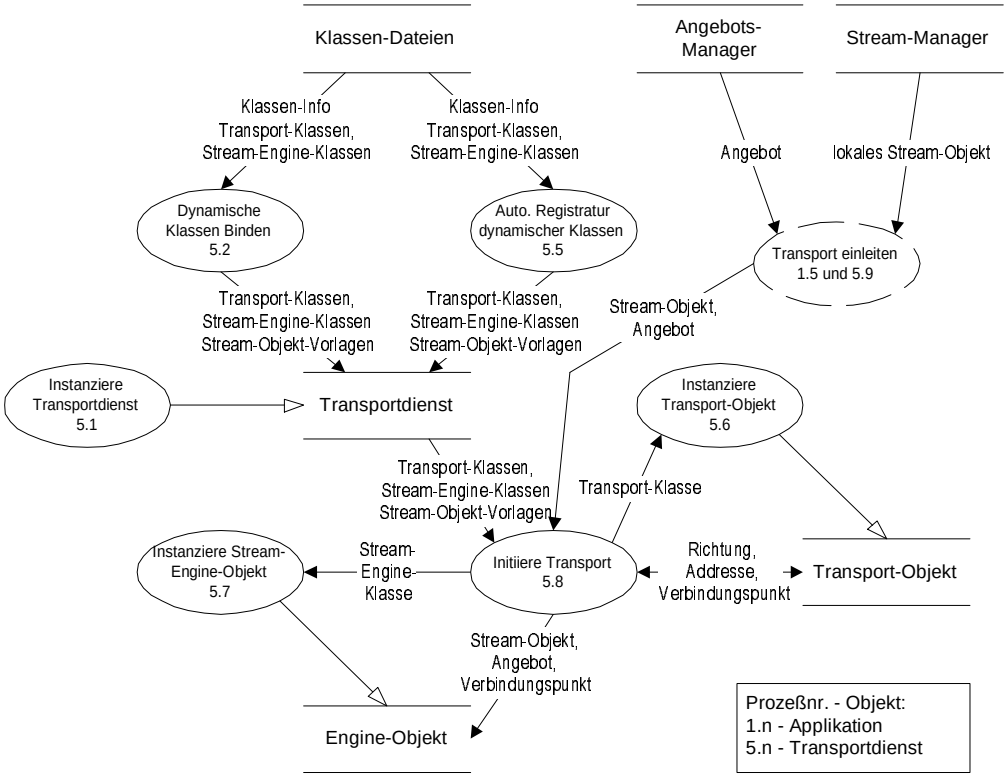

Abbildung 5.25 Datenflußplan des Transportdienstes

Teil II

Entwurf und Realisierung

6 Systementwurf

Während der Analyse wurde weitestgehend geklärt, *was* zu tun ist, unabhängig davon, *wie* es realisiert wird. Während des Entwurfs soll die Frage nach dem *wie* beantwortet werden. Die globale Problemlösungsstrategie für die Implementierung des Systems soll entwickelt werden. Der Systementwurf umfaßt Entscheidungen über die Organisation des Systems in Teilsysteme sowie wichtige Entscheidungen zur Konzeption und Vorgehensweise, die den Rahmen für die Implementierung bilden.

Aufgrund möglicher Konsequenzen muß während des Entwurfs darauf geachtet werden, daß das DMSM-System vornehmlich für NT-Workstations und in der Sprache C++ implementiert werden soll und als Kommunikationsprotokolle der DMSMS-Komponenten die IP-basierten Protokolle eingesetzt werden sollen.

6.1 Systemarchitektur

Wollte man das DMSM-System mit einem der verbreiteten Architekturkonzepte klassifizieren, so wäre das Konzept der *interaktiven Schnittstelle* das passende – ein System, in dem Interaktionen zwischen dem System und externen Agenten wie Anwendern, Geräten oder anderen Programmen überwiegen (siehe auch Abbildung 3.2). Die externen Agenten sind auf der einen Seite die Applikationen, die von einem Anwender direkt oder indirekt gesteuert werden, auf der anderen Seite sind es weitere DMSMS-Komponenten, die wiederum von Anwendern direkt bzw. indirekt gesteuert werden. Zu den wichtigsten Fragen im Zusammenhang mit interaktiven Schnittstellen gehören das Kommunikationsprotokoll zwischen den Systemkomponenten und den Applikationen, der Kontrollfluß innerhalb des Systems, der reibungslose Betrieb, die Fehlerbehandlung und die Eindämmung der Folgen unvorhersehbarer Betriebszustände auf Grund der möglichen Fernwirkung.

Alle Fragen befriedigend zu beantworten ist normalerweise ein iterativer Prozeß mit mehr als einem Durchlauf. Im Rahmen dieser Arbeit kann aber nur ein Iterationsschritt durchgeführt werden. Die folgenden Ausführungen haben somit nicht den Anspruch, die gestellten Fragen bis ins letzte Detail zu beantworten.

6.1.1 Partitionierung des Systems

Im ersten Schritt soll das DMSM-System in eine kleine Zahl von Teilsystemen partitioniert werden. Jedes Teilsystem umfaßt Aspekte des DMSM-Systems mit einer gemeinsamen Eigenschaft, z.B. ähnlicher Funktionalität. Ein Teilsystem ist weder ein Objekt noch eine Funktion, sondern ein Paket miteinander verbundener Klassen, Assoziationen, Operationen, Ereignisse und Einschränkungen mit einer wohldefinierten und kompakten Schnittstelle zu anderen Teilsystemen. Da ein Teilsystem in der Regel durch seine Dienste identifiziert wird, bietet sich die während der Analyse vorgenommene Einteilung in Dienste an.

Die Dienste, so wie sie im Objektmodell bisher auftauchten, sind dann als reine Schnittstellenobjekte zu sehen. Sie leiten den Kontrollfluß an die einzelnen Objektinstanzen innerhalb des eigenen Teilsystems weiter. Kontrollflüsse, die aus einem Teilsystem in die Anwendung führen, müssen in jedem Fall ebenfalls über diese Schnittstelle geführt werden. Ereignisse, die innerhalb einer Objektinstanz auftreten, dürfen somit nie direkt an die Anwendung übermittelt werden. Andernfalls würde der Weg zur Prozeßparallelisierung immens erschwert werden.

Dynamische Objekte werden ebenfalls als Teilsysteme betrachtet und den essentiellen Diensten gleichgestellt. Eine Anwendung kann die Schnittstelle zu einem dynamischen Objekt direkt bedienen.

Abbildung 6.1 zeigt eine mögliche Softwarestruktur. Angestrebtes Ziel sollte sein, den horizontalen Kontrollfluß weitestgehend zu vermeiden, und wenn er denn stattfinden muß, ihn ebenfalls über die Schnittstellen der einzelnen Teilsystem zu führen. Das DMSM-Basissystem soll helfen, dieses Ziel leichter zu erreichen. Allgemeine Klassen und globale Ressourcen können im Basissystem untergebracht werden.

Es wird hier festgelegt, daß eine DMSMS-Komponente immer über das Basissystem verfügen muß; auch die dynamischen Objekte können sich auf die Existenz des Basissystems verlassen, nicht aber auf die Existenz der vier angegebenen Dienste. Als positive Konsequenz für den Entwickler eines dynamischen Objektes ergibt sich damit, daß er lediglich partielle Kenntnisse über das Basissystem besitzen muß.

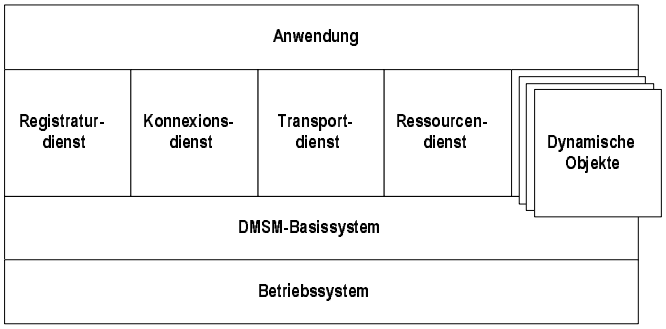

Abbildung 6.1 Softwarestruktur einer DMSMS-Komponente

Sobald der Kontrollfluß bis zu einer anderen DMSMS-Komponente führt, sollte dies nur innerhalb der gleichen Dienstkategorie möglich sein. Der Konnexionsdienst der Komponente A sollte z.B. nicht direkt mit dem Transportdienst der Komponente B kommunizieren. Außerdem sollte für die Kommunikation zwischen den einzelnen Dienstkategorien jeweils ein eigener Kanal benutzt werden. Einerseits wird das aufwendige Multiplexen bzw. Routen des Kontrollflusses vermieden, andererseits kann durch die Entkopplung eine einfachere Prozeßparallelisierung durchgeführt werden, die zugleich den positiven Nebeneffekt hat, daß blockierende Prozesse nicht eine ganze DMSMS-Komponente zum Erliegen bringen können.

6.2 Parallelitäten

Die inhärenten Parallelitäten[11], die im Analysemodell bestehen, sollten möglichst bewahrt werden. Sie fördern die Reaktionsfähigkeit des Systems und den flüssigen Ablauf komplexer Vorgänge innerhalb des gesamten DMSM-Systems.

Ein schematisches Vorgehen bei der Spezifikation paralleler Vorgänge besteht z.B. darin, den Kontrollfluß in VKN-Notation[12] darzustellen. Solange der kritische Pfad[13] von unbestimmter Dauer ist, muß versucht werden, die Vorgänge, die zu der unbestimmten Dauer führen, parallel auszuführen. Pfade, die eine Synchronisation mit dem dann parallel geführten Pfad erfordern, müssen ebenfalls parallel geführt werden.

Ein Beispiel ist die Veröffentlichung der Fähigkeiten einer DMSMS-Komponente. Der Vorgang könnte wie in Abbildung 6.2 dargestellt ablaufen.

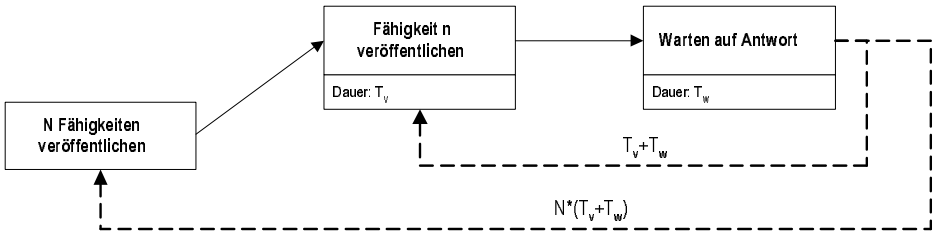

Abbildung 6.2 Veröffentlichung der Fähigkeiten ohne parallelem Prozeß

Zum Zeitpunkt der Veröffentlichung ist nicht bekannt, ob überhaupt eine Antwort eintreffen wird. Die Dauer T_w ist damit unbestimmt. Das Warten auf eine Antwort sollte deshalb parallel ablaufen. Der Vorgang könnte dann wie in Abbildung 6.3 dargestellt ablaufen.

[11] Zwei Objekte sind inhärent parallel, wenn sie gleichzeitig Ereignisse empfangen können, ohne miteinander zu interagieren.

[12] VKN, Vorgang-Knoten-Netz, DIN 69900: Alle Kästen sind Vorgänge, notiert mit ihren Zeiten, alle Pfeile sind Verknüpfungen; ein Folge-Vorgang wird erst dann ausgeführt, wenn alle Vorgänge, die mit ihm verknüpft sind, abgeschlossen sind.

[13] Der kritische Pfad ist der Pfad mit maximaler Dauer, der Anfang und Ende des Zyklus verbindet.

Für die Anwendung ist der Vorgang beendet, sobald alle Fähigkeiten veröffentlicht wurden. Auf die Antwort wird nicht gewartet. Einzige Bedingung für die Parallelisierung ist, daß der parallele Prozeß eine asynchrone Benachrichtigung der Anwendung durchführen kann. In vielen Fällen reicht es aus, wenn die Anwendung den Zustand eines parallelen Prozesses abfragen kann. So z.B. im vorherigen Beispiel: Der parallele Prozeß konfiguriert nach dem Eintreffen der Antwort den Konnexionslayer und beendet sich. Wenn die Anwendung beispielsweise mit einem Benutzer interagiert, wird der erste Schritt sein, eine Anfrage zu stellen. Die Anwendung hat erst dann ein Interesse, den Zustand des parallel abgesetzten Vorganges zu erfahren, wenn der Benutzer ausdrücklich eine Anfrage stellen will.

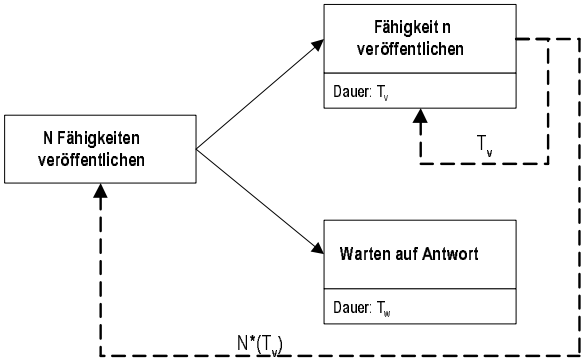

Abbildung 6.3 Veröffentlichung der Fähigkeiten mit parallelem Prozeß

Der Ablauf einer Anfrage und das Warten auf Anworten kann komplett parallelisiert werden. Der Vorgang der Anfrage wäre für die Anwendung sofort abgeschlossen, nachdem alle Anfrageobjekte instanziiert wurden. Der Vorteil einer solchen Implementierung bestünde in der hohen Verarbeitungsrate parallel eintreffender Antworten. Eine Entzerrung der Leistungsspitzen, wie sie in Kapitel 5.4.2 für die Senderseite vorgeschlagen wurde, ist dann nicht notwendig, sofern ein Sender auf den erfolgreichen Verbindungsaufbau wartet.

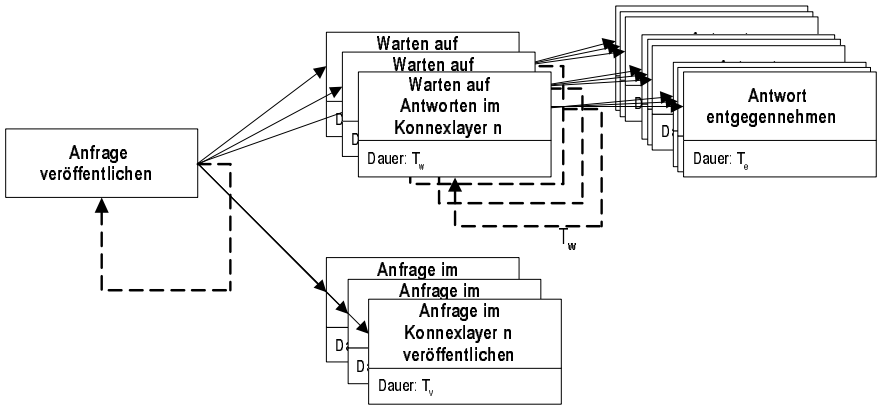

Abbildung 6.4 Parallele Vorgänge bei einer Anfrage

Die bisher angesprochenen Vorgänge sollten für einen reibungslosen Betrieb in jedem Fall wie vorgeschlagen implementiert werden. Darüber hinaus sind sämtliche Ereignisse, die während der gesamten Laufzeit einer Anwendung asynchron auftreten können, als parallele Prozesse zu implementieren. Dies sind beispielsweise der Empfang von veröffentlichten Fähigkeiten, Anfragen und Angeboten.

Die Entscheidung für oder wider einer Implementierung weiterer Parallelitäten, die über die oben genannten hinaus gehen, sollte nur durch Abwägen von Aufwand und Nutzen gefällt werden. Es ist zu bedenken, daß jeder parallele Vorgang ein nicht zu vernachlässigenden Verwaltungsaufwand sowohl im Betriebsystem als auch im DMSM-System erfordert.

6.3 Objektkommunikation

Innerhalb des DMSM-Systems kann die Objektkommunikation in drei Wirkungsbereiche eingeteilt werden. Die Kommunikation innerhalb bzw. zwischen den einzelnen Diensten, die Kommunikation zwischen Applikation und Diensten, und die Kommunikation zwischen den einzelnen DMSMS-Komponenten. Die Bezeichnung *Dienste* schließt in diesem Zusammenhang auch die dynamischen Objekte ein.

Je nach Wirkungsbereich und je nach Richtung des Kontrollflusses sollten unterschiedliche Kommunikationsmethoden eingesetzt werden. Die möglichen Kommunikationsmethoden unterscheiden sich hauptsächlich bzgl. Synchronität, Effizienz, Flexibilität und Robustheit.

Im Folgenden sollen einige Kommunikationsmethoden diskutiert werden, die zum Einsatz kommen könnten.

Direkter Methodenaufruf

Der direkte Methodenaufruf ist eine synchrone Kommunikationsmethode. Ein Methodenaufruf kehrt erst dann zurück, wenn die Befehlssequenz abgearbeitet wurde. Die Befehlssequenz ist meistens abhängig vom Objektzustand, so daß die Bearbeitungszeit der Sequenz schwanken kann. Diese Kommunikationsmethode sollte überall dort eingesetzt werden, wo einerseits Effizienz gefordert ist und andererseits gewährleistet werden kann, daß die maximale Bearbeitungszeit gering ausfällt.

Die Schnittstelle der Dienste, bei der der Kontrollfluß in den Dienst führt, sollte diese Methode nutzen. Bei umgekehrtem Kontrollfluß ist diese Methode mit Vorsicht zu benutzen, zumindest wenn die Gefahr besteht, daß sich eine Blockierung der Methode negativ auf das Verhalten anderer Komponenten auswirkt. Dieser Fall erscheint im ersten Augenblick sehr konstruiert, ist aber schnell herbeizuführen. Man stelle sich z.B. die synchrone Protokollsequenz aus Kapitel 5.3.3 (*Initiierung eines Datentransportes*) vor. Auf der Seite des Initiators wird durch einen Methodenaufruf die Befehlssequenz, wie in Abbildung 5.20 dargestellt, abgearbeitet. Würde nun in der Gegenstelle ein unvorhergesehener Zustand eintreten, der die Befehlssequenz unterbricht, hätte dies auch auf der Seite des Initiators eine Blockade zur Folge.

Bei Methodenaufrufen von Objekten, die nicht direkter Teil des DMSM-System sind, wie Applikationen und dynamischen Objekten, muß immer davon ausgegangen werden, daß solche unvorhersehbaren Zustände eintreten können.

Die wichtigste Voraussetzung für den direkten Methodenaufruf ist, daß das aufrufende und das aufgerufene Objekt sich im selben Adreßraum befinden. Der Vorteil liegt auf der Hand: Komplexe Kontrollflüsse können einfach und effizient realisiert werden. Der Nachteil: Objekte, die nicht direkter Teil des DMSM-System sind, wie Applikationen und dynamische Objekte, stellen ein erhöhtes Sicherheitsrisiko dar.

Kommunikationmethoden des Fenstersystems

Die meisten Fenstersysteme arbeiten nach dem gleichen Prinzip. Es gibt die sogenannten Ereignisquellen Maus, Keyboard, Timer usw. Ein Ereignis, das durch eine dieser Quellen erzeugt wurde, wird zunächst in einer Nachrichtenschlange plaziert. Das Kernprogramm eines Fenstersystems entnimmt nun zyklisch die Nachrichten aus der Nachrichtenschlange und verarbeitet sie. Die Verarbeitung ordnet das Ereignis einem Fensterelement zu und leitet das Ereignis an den mit dem Fensterelement assoziierten Prozeß weiter. Der Prozeß implementiert meist ein Untersystem, das ebenso aus einer Nachrichtenschlange mit zyklischer Verarbeitung besteht. Die Verarbeitung innerhalb des Prozesses führt dann die Funktionsaufrufe in Abhängigkeit der Ereignisse durch.

Dadurch, daß die Fenstersysteme synthetische Ereignisse zulassen, sprich Ereignisse, die Prozesse künstlich in die Nachrichtenschlange einfügen, kann eine recht mächtige Kommunikationsmethode realisiert werden, sogar über Prozeßgrenzen hinweg.

Eine solche nachrichtenorientierte Kommunikation eignet sich besonders gut für den asynchronen Kontrollfluß, der aus den Diensten heraus in die Applikation führt. Der Dienst plaziert eine Nachricht in der Nachrichtenschlange der Applikation und kann sofort seine Arbeit fortführen, ohne auf die Bearbeitung der Nachricht zu warten.

Zwei Dinge sind jedoch zu bedenken:

Befindet sich der Dienst innerhalb des Prozeßraumes der Applikation und werden die Methoden der Dienste durch denselben Thread wie die Verarbeitungsmethode des Fenstersystems aufgerufen, ist die Benachrichtigung nicht wirklich asynchron.

Bei intensiver Nutzung dieser Kommunikationsmethode wird die Portierbarkeit zu anderen Systemplattformen erschwert. Das Prinzip der Methode ist zwar bei verschiedenen Betriebsystemen sehr ähnlich, nicht aber die Realisierung.

Gegenstands-Beobachter-Kommunikationsmethode

Die Gegenstands-Beobachter-Kommunikationsmethode ist bei den meisten Implementierungen ein indirekter Methodenaufruf. Das Gegenstandsobjekt implementiert eine Benachrichtigungsfunktion, bei dessen Aufruf in allen angemeldeten Beobachterobjekten eine zentrale Methode aufgerufen wird. Die Parameter, die der zentralen Methode übergeben werden, beschreiben das Ereignis und seine Quelle.

Abbildung 6.5 zeigt das Objektmodell für den Fall, daß die Gegenstands-Beobachter-Kommunikationsmethode für den Kontrollfluß Dienst → Applikation eingesetzt wird.

Der Vorteil gegenüber des direkten Methodenaufrufs liegt in der Flexibilität und Robustheit. Das beobachtete Objekt muß keinerlei Kenntnisse über die innere Struktur des Beobachters besitzen. Es liegt allein im Ermessen des Beobachters, ein Ereignis auszuwerten.

Wie die Assoziation zwischen Gegenstand und Beobachter implementiert wird, entscheidet darüber, ob diese Kommunikationsmethode synchron bzw. asynchron ist. Beispielsweise könnte die Assoziationsmethode ähnlich wie bei einem Fenstersystem implementiert werden oder man überführt den Kontrollfluß in eine serielle Sequenz und transportiert ihn mittels einer *Pipe*[14] über Prozeßgrenzen hinweg, womit dann eine leistungsfähige Interprozeßkommunikation realisiert wäre.

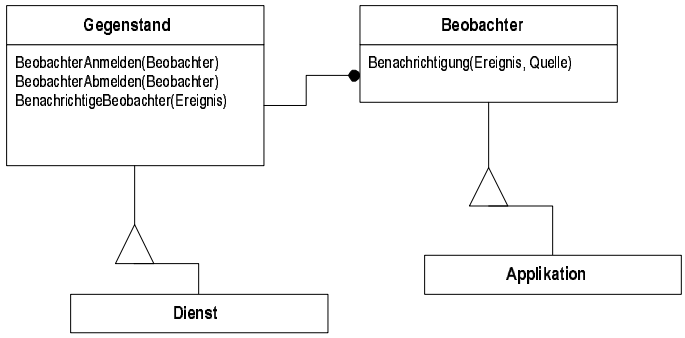

Abbildung 6.5 Prinzip der Gegenstands-Beobachter-Kommunikationsmethode

Serieller binärer Objekttransport

Begutachtet man die Datenflußdiagramme im Kapitel 5.4, so stellt man fest, daß es nur eine einzige synchrone Protokollsequenz gibt, nämlich die des Transportdienstes, während der Einleitung eines Datentransportes. Alle anderen Kontrollflüsse beschränken sich auf den Austausch von Objekten, wie z.B. Veröffentlichung einer Fähigkeit, eines Angebotes oder einer Anfrage.

Aus diesem Grund ist es naheliegend, die Objekte als solche zwischen den Komponenten zu übertragen. Das Problem, das sich einem dabei entgegenstellt, ist die Eigenschaft des Übertragungskanals.

Z.B. sollte bei 1:M-Beziehungen IP-Multicast eingesetzt werden. Dadurch ist man gezwungen, die Objektübertragung verbindungslos, durch Versenden von UDP-Paketen zu bewerkstelligen. Der Empfänger muß dabei die Größe der zu empfangenden Pakete kennen. Da die Größe der Objekte sehr unterschiedlich sein kann, muß ein Header kon-

[14] Ein Übertragungskanal mit zwei Endpunkten. Ein Prozeß mit Zugang zu einem Endpunkt kann über die Pipe mit einem Prozeß kommunizieren, der Zugang zum anderen Endpunkt besitzt.

stanter Größe einem Objekt vorangestellt[15] werden. Der Header gibt Auskunft über Größe und Klasse des angehängten Objektes. Die Daten, die dem Header folgen, repräsentieren das Objekt in serieller Form.

Es wäre hier sehr hilfreich, würde das Objekt selbst eine Methode bereitstellen, seine Attribute von der normalen Objektdarstellung in die serielle Darstellung zu überführen und umgekehrt. Es müßte nur berücksichtigt werden, daß je nach Objektzustand die Attribute von Instanz zu Instanz unterschiedlich lang sein können, so daß eine einfache Binärsequenz nicht ausreicht. Es müssen darüber hinaus für jedes Attribut Typinformationen und Größe mitgeliefert werden.

Desweiteren ist zu bedenken, daß die Objektgröße bei der Nutzung von UDP-Paketen begrenzt ist und daß die Transformationsvorschrift, ein C++-Objekt in seine serielle Binärdarstellung zu überführen, von der Rechnerarchitektur[16] abhängt.

Wird die Objektübertragung mit Hilfe einer TCP/IP-Verbindung durchgeführt, ergibt sich gegenüber UDP der Vorteil der unbegrenzten Objektgröße und der gesicherten Übertragung.

In vielen Bibliotheken wird die serielle Übertragung von C++-Objekten speziell für den Transport via TCP/IP-Verbindungen bereits angeboten. Tests haben gezeigt, daß ein Großteil der bereitgestellten Funktionalität auch für die Übertragung via UDP verwendet werden kann.

6.4 Dynamische Objekte

Dynamische Objekte statten eine DMSMS-Komponente mit Wissen und Können aus. Erst mit der Existenz dynamischer Objekte erhält sie ihre Fähigkeiten und kann mit anderen Komponenten in Beziehung treten.

Der Begriff „dynamisches Objekt“ meint im Zusammenhang mit dem DMSM-System Objekte, die deklaratives und prozedurales Wissen in einer bestimmten Programmiersprache implementieren und während der Laufzeit des Systems integriert und entfernt werden können. In ihnen ist das gesamte Spezialwissen untergebracht, das für einen Verbindungsaufbau bzw. Datentransport notwendig ist. Ihr innerer Aufbau fällt je nach Netztechnologie und Stream-Struktur sehr unterschiedlich aus.

Damit der Einsatz eines solchen Konzeptes möglich ist, muß das Betriebssystem das dynamische Binden von Binärobjektdateien unterstützen. In der Regel wird dies von allen modernen Betriebssysteme mit Hilfe der dynamischen Bibliotheken realisiert. Es können somit Codefragmente in binärer Form während der Laufzeit hinzugefügt und entfernt werden.

---

[15] Header und Objekt dürfen keinesfalls getrennt verschickt werden, denn es könnte sein, daß das Objekt eher eintrifft als der Header, oder eines der beiden verloren geht.

[16] Unterschiedliche Rechnerarchitekturen speichern Daten teilweise in unterschiedlicher Byte-Reihenfolge. Intel-basierte Computer speichern z.B. die Daten in umgekehrter Byte-Reihenfolge wie Macintosh-Computer (Motorola). Die Byte-Reihenfolge von Intel, *Little Endian* genannt, ist auch die Umkehrung der Netzwerkstandardreihenfolge *Big Endian*.

Fordert man nun, daß die aufrufende Instanz keine Kenntnisse über die innere Struktur des Codefragmentes bzw. des binären Objektes kennen muß, kann das Konzept der dynamischen Objekte realisiert werden. Diese Forderung muß aber gar nicht explizit gestellt werden, da sie konform mit der in der Problembeschreibung (Kapitel 5.1) gestellten Forderung ist: Die Steuerung des Stream-Transportes soll polymorph bzgl. normaler, verteilter und virtueller Stream-Objekte sein.

Mögliche Realisierung der Schnittstellen

Für die Implementierung der Schnittstellen eines dynamischen Objektes sind alle in Kapitel 6.3 vorgestellten Kommunikationsmethoden geeignet.

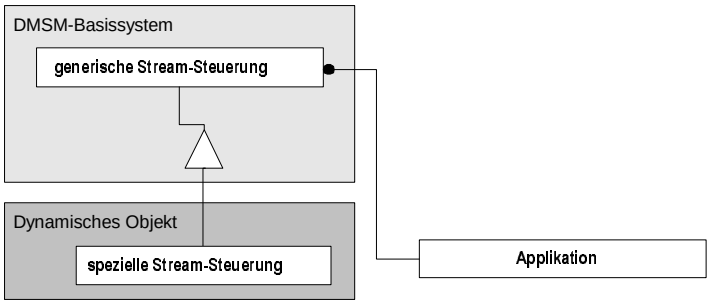

Abbildung 6.6 Direkter Methodenaufruf bei dynamischen Objekten

Bei Verwendung von C++ kann der direkte Methodenaufruf mit Hilfe der virtuellen Funktionen realisiert werden. Dazu wird die Schnittstelle über virtuelle Methoden formuliert und in einer Basisklasse des DMSM-Basissystems untergebracht. Jedes dynamische Objekt erbt die Schnittstelle und überschreibt die Methoden, die es implementiert. Die Anwendung würde dann immer die Schnittstelle der Basisklasse bedienen, ohne Kenntnisse über das eigentliche gesteuerte Objekt zu besitzen. Abbildung 6.6 zeigt das Objektmodell für ein dynamisches Stream-Engine-Objekt mit spezialisierter Stream-Steuerung.

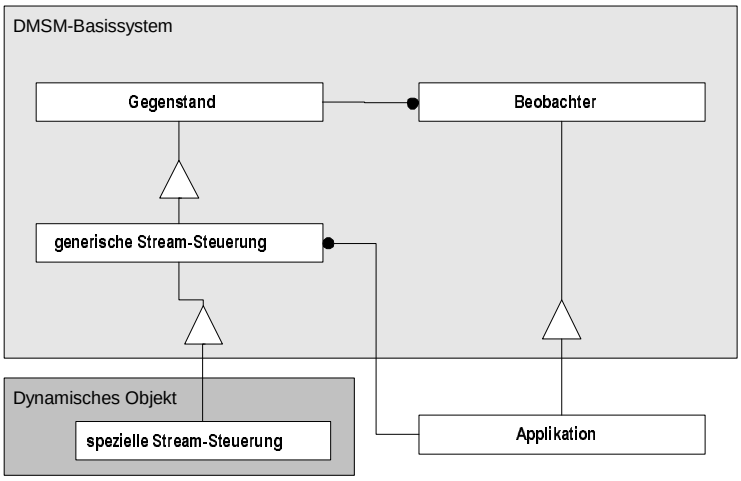

Abbildung 6.7 Bidirektionale Kommunikation mit dynamischen Objekten

Der direkte Methodenaufruf ist für den Kontrollfluß vom dynamischen Objekt in die Anwendung ungeeignet. Das dynamische Objekt besitzt keinerlei Kenntnisse über die Anwendung. In diesem Fall ist die Gegenstands-Beobachter-Kommunikationsmethode am besten geeignet. Für eine bidirektionale Kommunikation müssen deshalb beide Methoden kombiniert werden. Abbildung 6.7 zeigt die dafür notwendige Erweiterung des Objektmodells aus Abbildung 6.6.

Problematik nicht vorhersehbarer Objektzustände

Die eben vorgestellte Lösung birgt einige Gefahren in sich. Damit der direkte und indirekte Methodenaufruf möglich ist, muß sich das dynamische Objekt zusammen mit der Applikation im Adreßraum des DMSM-Basissystems befinden. Es kann damit durch unvorhersehbare Objektzustände eine ganze Komponente zum Erliegen bringen.

Entweder verläßt man sich auf die gewissenhafte Implementierung der dynamischen Objekte, oder man teilt jedem dynamischen Objekt einen eignen Prozeßraum zu. Letzeres würde die Implementierung einer Interprozeßkommunikation erfordern.

Für die Beispielimplementierung muß dieses Thema sicherlich nicht weiter diskutiert werden, für eine produktreife Implementierung ist die Trennung der Adreßräume dann vorzunehmen, wenn eine hohe Robustheit des Systems erzielt werden soll.

6.5 Behandlung von Grenzbedingungen

Ein Großteil der Entwurfsarbeit richtet sich auf das Verhalten im Normalzustand. Im folgenden sollen die Grenzbedingungen Initialisierung, Terminierung und Absturz betrachtet werden.

Initialisierung

Einerseits muß das DMSM-System im Ganzen und andererseits jede DMSMS-Komponente für sich von einem ruhenden Anfangszustand in einen dauerhaften Normalzustand gebracht werden.

Das größte Problem, das sich während der Initialisierung stellt, ist die inhärente Parallelität der Komponenten. Wenn man den Fall konstruiert, daß alle Komponenten zur gleichen Zeit aktiviert werden, stellt sich die Frage, ob das System sich genau so konfiguriert, als wenn alle Komponenten hintereinander aktiviert werden. Die Frage muß mit einem klaren *nein* beantwortet werde. Betrachtet man beispielsweise den Algorithmus für die Komponentenintegration aus Kapitel 5.3.1, zeigt sich, daß mit wachsender Zahl der Komponenten die Belastung des Netzes während der Integration proportional wächst. Durch die Möglichkeit der Clusterbildung wird es mit wachsender Komponentenzahl immer wahrscheinlicher, daß sich das System beim parallelen Start anders konfiguriert als beim sequentiellen.

Nun hat jede Komponente ihre individuellen Startzeiten, bedingt durch die gegebenen Toleranzen bei der Hardware inklusive ihrer Konfiguration und bei der Softwarekonfiguration. Man kann annehmen, daß diese Schwankungen im Sekundenbereich liegen.

Erreicht man bei der Implementierung der Komponentenintegration, daß die Dauer des Integrationsvorganges sehr viel kürzer ist als die mittlere Abweichung der einzelnen Startzeitpunkte, so kann ein paralleler Start kaum erzielt werden. Sollte es dennoch vorkommen, daß die natürlich gegebenen Abweichungen zu klein ausfallen, kann eine künstliche Variation das Problem entschärfen.

Ein weiteres Problem ergibt sich, wenn man bestimmte Abläufe vom Zustand des gesamten DMSM-Systems abhängig machen wollte. Der Grund dafür ist die fehlende Kenntnis einer Komponente über den Sollzustand des Systems. Ein Beispiel soll das Problem verdeutlichen: Es soll ein System realisiert werden, das nach dem Einschalten automatisch den Datenstrom einer Überwachungskamera aufzeichnet. Die Aufzeichnung soll mit Hilfe eines parallel verteilten Stream-Objektes erfolgen und jedes Teil-Stream-Objekt befindet sich auf einer anderen Komponente. Die Aufzeichnung soll erst dann starten, wenn alle Komponenten bereit sind.

Das Szenario könnte in etwa wie folgt ablaufen: Das System wird aktiviert. Alle Komponenten befinden sich in der Initialisierungsphase. Die Videoencoderkomponente sucht im Anschluß an die eigene Initialisierung nach Aufzeichnungsobjekten.

Wann soll die Aufzeichnung beginnen, wenn die Videoencoderkomponente vorher nicht weiß, aus wieviel Teilen das Aufzeichnungsobjekt besteht? Die Frage kann nicht klar beantwortet werden.

Es handelt sich hierbei um ein konzeptionelles Dilemma. Es bleibt nicht anderes übrig, als die Videoencoderkomponente mit a priori Wissen über die Systemtopologie auszustatten. Im Beispiel bedeutet dies konkret, daß die Videoencoderkomponente die Kenntnis über die Anzahl der Bestandteile des Aufzeichnungsobjektes besitzen muß.

Eine Beseitigung des Dilemmas hätte eine starke Einschränkung der Flexibilität zur Folge. Deshalb sollte das Problem in Kauf genommen werden, zumal es nur in wenigen Situationen eine Rolle spielt.

Terminierung

Eine Terminierung des gesamten DMSM-Systems als solche gibt es nicht. Das DMSM-System ist beendet, wenn die letzte im System existierende DMSMS-Komponente beendet wurde.

Jede Komponente sollte mit einer Abmeldung im gesamten DMSM-System die Terminierung einleiten. Dazu veröffentlicht sie das Vorhaben über die Basisadresse. Die Komponente darf nicht verpflichtet werden, Reaktionen anderer Komponenten auf das angekündigte Vorhaben verarbeiten zu müssen. Sie kann sofort die Empfangsbereitschaft einstellen. Weiterhin darf nicht damit gerechnet werden, daß jede Komponente die Ankündigung der Terminierung empfängt. Dennoch muß gewährleistet sein, daß alle belegten Ressourcen bei fremden Komponenten freigegeben werden. Dies kann beispielsweise aus dem Kontext der Steuerungsprotokolle ersehen werden.

Absturz

Ein Absturz ist die ungeplante Terminierung einer DMSMS-Komponente, die durch Benutzerfehler, Programmfehler, erschöpfte Systemressourcen oder einen Stromausfall verursacht sein kann.

Wichtig ist, daß bei dem Absturz einer Komponente andere Komponenten nicht in ihrem Betrieb beeinträchtigt werden. Außerdem müssen die von der abgestürzten Komponente belegten Ressourcen freigegeben werden.

Es ist für eine Komponente nicht immer eindeutig erkennbar, wann die von einer fremden Komponente belegten Ressourcen nicht mehr benötigt werden. Im normalen Betrieb geht die Freigabe aus dem Kontext der Protokolle hervor, bei einem Absturz kann dieser Kontext meistens nicht mehr hergestellt werden. Deshalb sollte die Ressourcenvergabe für externe Komponenten einem übergeordneten Kontext zugeordnet werden.

Beim DMSM-System werden Ressourcen fremder Komponenten nur während der Datenübertragung belegt. Während der Datenübertragung sind in den meisten Fällen dezidierte Steuerungs- und Übertragungsverbindungen vorhanden, deren Verwaltung vom Betriebssystem durchgeführt wird. Der Absturz einer Komponente wird in der Regel vom Betriebssystem registriert, so daß zumindest die Verbindungen ordnungsgemäß beendet werden. Es bietet sich somit an, die belegten Ressourcen einer fremden Komponente zusätzlich dem Kontext der Verbindungen zuzuordnen und nicht alleine dem Kontext des Protokolls.

7 Beispielimplementierung

Ziel der Beispielimplementierung ist es in erster Linie die Möglichkeit zu bieten, die Analyse und den Entwurf zu evaluieren. Das Ergebnis der Evaluation dient der Verfeinerung der Modelle in einem neuen Iterationsschritt des Entwurfs. Viele Ansätze der Analyse sind rein theoretischer Natur und müssen sich erst in der Praxis unter realen Betriebsbedingungen bewähren. Für diesen Zweck sollen nur die grundlegenden Konzepte umgesetzt werden.

Damit der Umfang der Implementierung dem Rahmen dieser Arbeit gerecht wird, soll der Ressourcendienst nicht Teil der Beispielimplementierung sein.

Die Beispielimplementierung wird für das Betriebssystem Windows-NT durchgeführt. Als Entwicklungswerkzeug wird Visual C++ (Professional Edition) eingesetzt und für die Kommunikationsprotokolle wird UDP bzw. TCP verwendet.

Es wird zunächst das DMSM-System implementiert. Darauf aufbauend wird ein Demonstrationssystem entwickelt, mit dessen Hilfe die Evaluation aller Systemfunktionalitäten möglich wird.

7.1 Objektentwurf und Realisierung

Der Objektentwurf verfeinert das während der Analyse erarbeitete Objektmodell. Es werden ggf. neue Objekte eingeführt, um komplexere Objekte in weniger komplexe zu partitionieren. Aus dem dynamischen Modell werden die Operationen konkretisiert und ggf. Algorithmen spezifiziert.

Für alle zu implementierenden Klassen, Attribute und Methoden werden im Quellcode englische Bezeichnungen benutzt, die sich aus der wörtlichen bzw. sinngleichen Übersetzung ergeben.

In den folgenden OMT-Diagrammen sind die Klassen nicht vollständig abgebildet. Für die genaue Syntax, Typinformationen sowie die Deklaration aller Methoden und Attribute sei auf die Headerdateien des Quellcodes verwiesen. Alle Klassen in den OMT-Diagrammen, denen ein Stern (*) vorangestellt ist, stammen aus den Basisklassen der Entwicklungsumgebung.

7.1.1 Das Basissystem *DMSMSCore*

Für die Aufnahme einer Klasse in das Basissystem *DMSMSCore* wurden folgende Regeln angewendet:

- Es darf keine Abhängigkeit von einem der Dienste bzw. dynamischen Objekte bestehen.
- Die erste Regel impliziert die zweite: Sobald eine Klasse in mehr als einem Dienst bzw. dynamischen Objekt eingesetzt wird, muß sie im Basissystem implementiert werden.

Hilfsklassen

Die Basisklassen *ProtocolType*, *ApplicationType*, *MediaType* und *DataDirection* dienen ausschließlich der Definition der entsprechenden Datentypen und implementieren keinerlei Funktionalität.

Die Klasse *Capability* deklariert den Datentyp *Fähigkeit* und implementiert mit der Methode *IsSociable()* die Prüfung auf Verträglichkeit wie sie in Kapitel 5.2.3 definiert wurde.

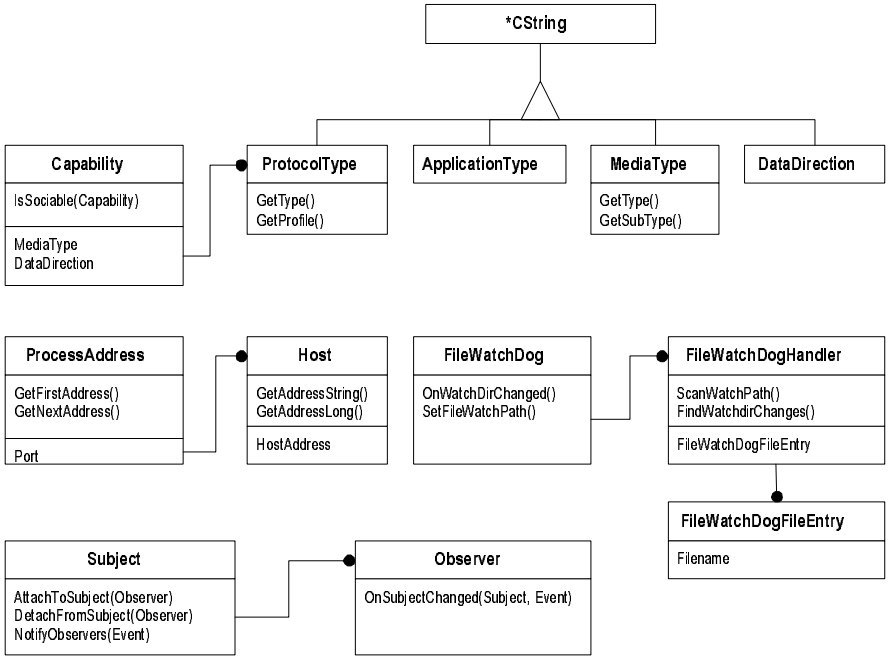

Abbildung 7.1 Objektmodell der Hilfsklassen

Mit Hilfe der Klasse *ProcessAddress* werden Prozesse auf Rechnern innerhalb des DMSM-Systems adressiert. Die Klasse ist für *Multihomed-Hosts* konzipiert. Dies bedeutet, daß ein Rechner durchaus über mehrere Netzwerkkarten und damit über zwei IP-Adressen erreichbar sein kann. Fehlende Argumente bei der Initialisierung führen zur Ermittlung der eigenen Rechneradressen.

Die Klassen *Subject* und *Observer* implementieren die Gegenstands-Beobachter-Kommunikationsmethode. Ein zu beobachtender Gegenstand leitet seine Klasse von *Subject* ab, ein Beobachter leitet seine Klasse von *Observer* ab und überschreibt die virtuelle Funktion *OnSubjectChanged()*. Damit die Kommunikation aktiviert wird, ruft der Beobachter *AttachToSubject(...)* des Gegenstandes auf. Jedes Ereignis, das signalisiert werden soll, definiert der Gegenstand selbst. Um ein Ereignis zu signalisieren, muß der Gegenstand stets die Methode *NotifyObservers(...)* aufrufen. Mittels der Funktion *DetachFromSubject(...)* kann die Kommunikationsverbindung unterbrochen werden.

Die Klassen, deren Name mit *FileWatchDog* beginnen, implementieren einen Hilfsdienst. Ein Objekt, das von der Klasse *FileWatchDog* abgeleitet wurde, erbt die Fähigkeit, ein oder mehrere Pfade im Dateisystem zu überwachen. Der Dienst kann ganze Verzeichnisbäume überwachen und läßt sich auf bestimmte Dateitypen spezialisieren. Sobald eine Änderung in einem der überwachten Verzeichnisbäume eintritt, wird die Ursache über die virtuelle Funktion *OnWatchDirChanged(...)* signalisiert. Diese Basisklasse wird für die automatische Registrierung von dynamischen Objekten und explizit assoziierten Stream-Objekten eingesetzt.

Stream-Klassen

StreamObject ist die zentrale Klasse der Stream-Klassen. Die Ableitung von der Klasse *CDocument* ermöglicht die einfache Realisierung eines Stream-Objekt-Editors mit Hilfe einer Standard Multi-Document-Interface-Applikation[17]. Die Realisierung des Editors beschränkt sich dadurch auf den Entwurf der Bildschirmmaske. Ein solcher Editor wurde im Rahmen dieser Arbeit implementiert. Er dient der Bearbeitung und Überprüfung der mit einem Stream-Objekt gespeicherten Metadaten.

Die Funktion *Serialize(..., DataDirection)* überführt das Stream-Objekt in Abhängigkeit der Datenrichtung in einen seriellen binären Datenstrom oder umgekehrt. Mit ihr läßt sich eine Instanz der Klasse *StreamObject* über eine Netzverbindung versenden bzw. empfangen, oder in einer Datei speichern bzw. aus einer Datei lesen.

Die Klassen *StreamObjects* und *Announcements* sind reine Referenzlisten. Die Klassen *DMStreamObject* und *DMAnnouncement* erweitern die Referenzlisten um spezielle Bearbeitungsmethoden.

Die Klasse *StreamObjectTemplate* und *Announcement* erweitern die Klasse *StreamObject*. Das *StreamObjectTemplate* kann von einem Stream-Engine-Objekt ausgefüllt werden, womit es festlegen kann, wieviel Instanzen auf einer Komponente existieren dürfen, ob es ein verteiltes Stream-Objekt sein kann und ob die Objektdaten explizit

[17] Vorlagen für MDI-Applikationen sind Teil der Microsoft-Foundation-Class-Library

bzw. implizit assoziiert werden. Bei expliziter Assoziation erben alle aus dem *StreamObjectTemplate* gebildeten Instanzen die Werte aller Attribute seiner Basisklasse.

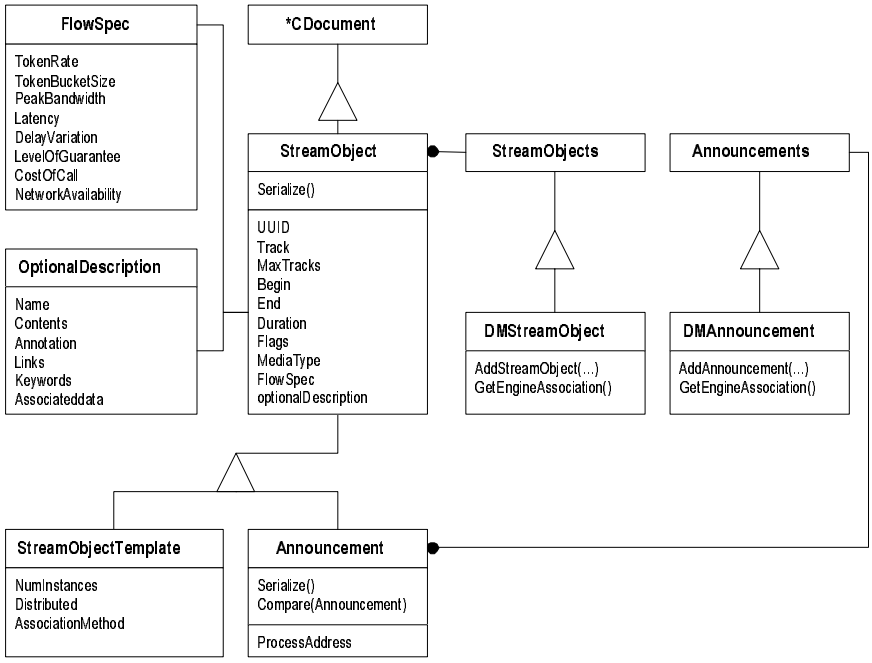

Abbildung 7.2 Objektmodell der Stream-Klassen

Verwaltung dynamischer Objekte

Die Verwaltung dynamischer Objekte gliedert sich in zwei Aufgabenbereiche. Der *ClassBroker* verwaltet ausschließlich Informationen über die verfügbaren Klassen dynamischer Objekte. Der *ObjectBroker* verwaltet die Instanzen dynamischer Objekte.

Der *ClassBroker* kann über die Methode *LoadDLLModule(...)* angewiesen werden, eine dynamische Bibliothek zu analysieren und, falls sie ein dynamisches Objekt enthält, die notwendigen Informationen zu extrahieren. Dazu wird die Bibliothek geladen und geprüft, ob sie eine Funktion mit dem Namen *DMSMSClassInfo(...)* exportiert. Ist das der Fall, wird die Funktion aufgerufen. Mit den von der Funktion zurückgelieferten Informationen werden die Attribute in einer *ClassInfo*-Instanz initialisiert. Außerdem liefert die Funktion eine Referenz auf eine *CRuntimeClass*-Struktur. In dieser Struktur sind Typinformationen abgelegt, die während der Laufzeit eine exakte Auskunft über den Typ der Basisklasse und der eigenen Klasse zuläßt. Desweiteren ist in der Struktur die Adresse des Konstruktors abgelegt, so daß für die Instanziierung der Klasse nur die Adresse angesprungen werden muß.

Zur Sicherheit sollte vor dem Aufruf des Konstruktors geprüft werden, ob die Basisklasse einer der Klassen *GenericStreamControl* bzw. *GenericStreamTransport* ent-

spricht. Denn deren Basisklasse ist *DynamicObject* und damit *CRuntimeClass*, was die Gültigkeit der Referenz auf die *CRuntimeClass*-Struktur gewährleistet.

Die Struktur *CRuntimeClass* wird von der Entwicklungsumgebung bereitgestellt. Sollte das System auf ein Nicht-Windows-System portiert werden, kann alternativ eine eigene Klasse implementiert werden, die die von den meisten C++-Compilern mittlerweile standardmäßig unterstützten RTTI-Informationen[18] nutzt.

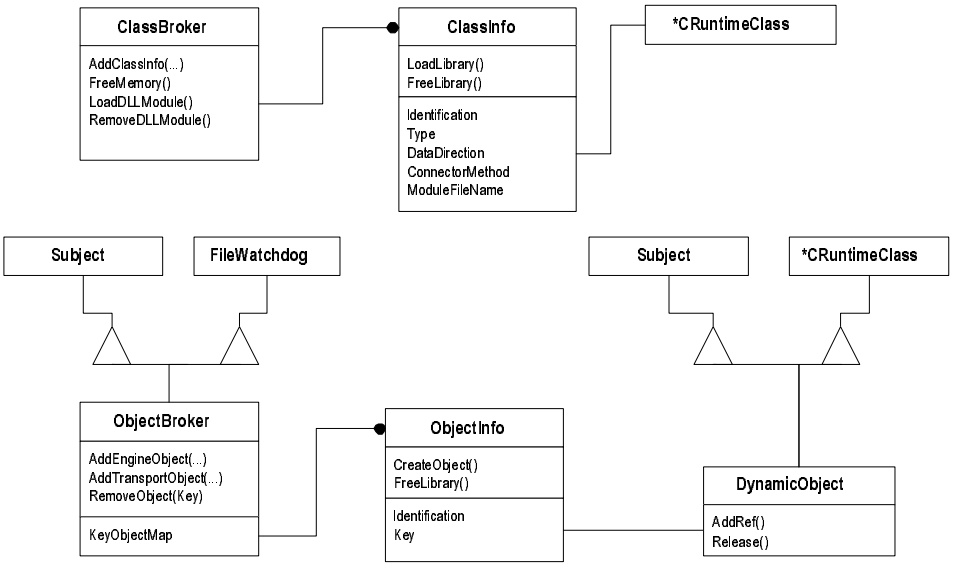

Abbildung 7.3 Objektmodell der Verwaltung dynamischer Objekte

Nach der Analyse der dynamischen Bibliothek wird sie wieder freigegeben. Sie wird erst wieder geladen, wenn eine Instanz der in ihr abgelegten Klasse gebildet werden soll. Für die Instanziierung der dynamischen Objekte ist der *ObjectBroker* verantwortlich.

Die Instanz eines dynamischen Objektes kann durch Aufruf einer der *Add*-Methode des *ObjectBrokers* erzeugt werden. Der *ObjectBroker* erhält dazu die Identifikation und erzeugt eine *ObjectInfo*-Instanz. Diese wird benötigt, um die Freigabe der Bibliothek zu koordinieren; sobald die letzte Instanz eines Objektes aus einer Bibliothek aufgegeben wurde, wird die Bibliothek freigegeben. Ergebnis der *Add*-Funktion ist die Adresse der Instanz eines dynamischen Objektes.

Der *ObjectBroker* wird von *Subject* und *FileWatchDog* abgeleitet. Als *FileWatchDog* registriert er Änderungen im Modulverzeichnis und kann den *ClassBroker* ggf. anweisen, die *ClassInfo*-Liste anzupassen. Als *Subject* kann der *ObjectBroker* Interessenten die Änderungen in der Modulkonfiguration mitteilen.

[18] RunTime-Type-Information

Durch die Ableitung der Klasse *DynamicObject* von *Subject* wird lediglich die Gegenstand-Beobachter-Kommunikationsmethode als Standardmethode vorgeschlagen.

Stream-Steuerung

Für die Einleitung und Steuerung des Datentransportes stehen zwei Schnittstellen zur Verfügung. Dynamische Objekte, die das Wissen über den Aufbau von Transportverbindungen implementieren, benutzen als Basisklasse *GenericStreamTransport*, und solche, die das Wissen über die Steuerung des Stream-Transportes implementieren, leiten ihre Klassen von *GenericStreamControl* ab. Einerseits können die Klassen damit als dynamische Objekte verwaltet werden und andererseits erben sie Basisfunktionalitäten, die die Implementierung stark vereinfachen können.

Die virtuellen Funktionen der Basisklasse *GenericStreamTransport* müssen allesamt überschrieben werden. Die Methode *HowToCallMe(...)* muß die Struktur und die Länge der eigenen Adresse liefern. Die Adresse ist so zu formulieren, daß eine externe Gegenstelle, die über das gleiche Transportobjekt verfügt, die eigene Komponente über diese Adresse erreichen kann. Es reicht somit aus, daß ein Transportobjekt seine Adreßdarstellung selbst interpretieren kann. Die Methode *TransportTo(..., DataDirection)* muß den Verbindungsaufbau einleiten. Sie erhält als Argumente die Adresse der Gegenstelle und die Datenrichtung. Wenn der Verbindungsendpunkt gültig ist (siehe auch Kapitel 0), signalisiert die Methode mit ihrem Rückgabewert einen erfolgreichen Verbindungsaufbau. Der Verbindungsendpunkt muß über die Methode *GetConnectionPoint(...)* exportiert werden können. Die Methode *TransportTo(...)* kann synchron implementiert werden, weil die Transportdienste der zu verbindenden Gegenstellen während des Verbindungsaufbaus synchronisiert sind.

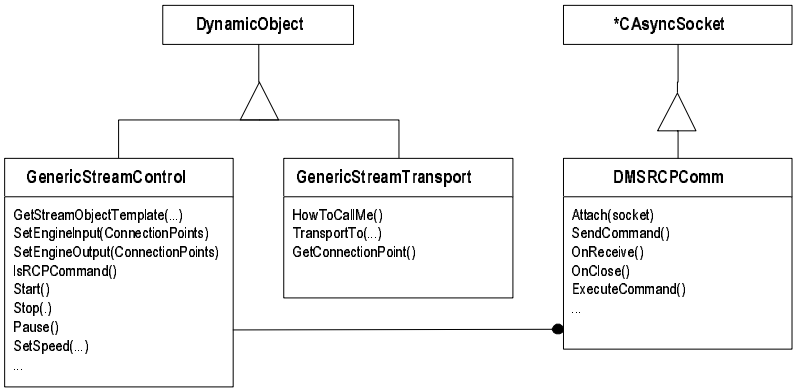

Abbildung 7.4 Objektmodell der Stream-Steuerung

Für jeden Teil-Stream eines DM-Streams wird ein Transportobjekt instanziiert. Die Verbindungsendpunkte der einzelnen Transportobjekte werden in einem Array gesammelt und anschließend der Stream-Engine-Instanz übergeben.

Ein Stream-Engine-Objekt sollte von *GenericStreamControl* abgeleitet werden, damit es die Beispielimplementierung des virtuellen Steuerungsbusses nutzen kann.

Außer den Methoden *SetEngineInput(...)* und *SetEngineOutput()* muß eine Klasse, die von der Basisklasse *GenericStreamControl* abgeleitet wurde, keine weiteren virtuellen Methoden zwingend überschreiben. Die Methode *GetStreamObjectTemplate(...)* sollte implementiert werden, wenn z.B. die Zuordnung explizit assoziierter Stream-Daten automatisch vom Registraturdienst durchgeführt werden soll, oder wenn die automatische Spurvergabe verteilter Aufzeichnungsobjekte genutzt werden soll.

Die Steuerkommandos wie *Start(), Stop(), SetSpeed()* usw. der Basisklasse *GenericStreamControl* müssen nur überschrieben werden, wenn einerseits der virtuelle Steuerungsbus des DMSM-Systems genutzt werden soll, und anderseits die Stream-Engine die entsprechende Funktionalität überhaupt realisiert.

Der virtuelle Steuerungsbus wird durch die Klasse *DMSRCPComm*[19] implementiert. Für jede Stream-Engine-Instanz der Gegenstellen, wird eine *DMSRCPComm*-Instanz erzeugt. Die *DMSRCPComm*-Instanzen werden über das *Attach(socket)* Kommando direkt an die vom Transportdienst bereits genutzten Kommunikationsverbindungen gebunden. Es ist damit kein weiterer Verbindungsaufbau notwendig. Die *DMSRCPComm*-Instanzen sind so verknüpft, daß die Steuerungskommandos der Wurzel einer 1:M-Beziehung in allen Blättern ausgeführt wird, und daß ein Steuerungskommando eines Blattes nur bei der Wurzel ausgeführt wird. Jede Instanz kann über die Methode *IsRCPCommand()* feststellen, ob das Kommando lokal oder extern erzeugt wurde.

Die genaue Spezifikation des DMSRCP-Protokolls kann der Datei *DMSRCProtocol.h* entnommen werden.

7.1.2 Der Konnexionsdienst *KonnexService*

Einer Applikation bleibt die innere Struktur des *KonnexService* verborgen. Sie bedient lediglich die Methoden der Klasse *KonnexService*.

Mit Hilfe der statischen Methode *KonnexService*::*Instance()* kann die Applikation die Adresse der Schnittstelle erfahren. Die Adresse ist während der gesamten Laufzeit der Applikation gültig. Erst mit Aufruf der Methode *Destroy()* wird der Dienst beendet und die Adresse ungültig.

Falls der Konnexionsdienst vor dem ersten Aufruf der Methode *KonnexService*::*Instance()* noch nicht initialisiert ist, werden vor der Rückkehr der Methode alle notwendigen Instanzen erzeugt und die Methode *AnnounceMyCapabilities()* automatisch aufgerufen. Die Methode *AnnounceMyCapabilities()* implementiert den Algorithmus zur Komponentenintegration exakt nach der Definition in Kapitel 5.3.1. Bedingt durch die inhärenten Parallelitäten der beteiligten Objekte ist die Implementierung des

[19] DMSRCP steht für Distributed Media Stream Remote Control Protocol

Algorithmus nur schwer nachvollziehbar und soll deshalb im folgenden noch einmal grob skizziert werden.

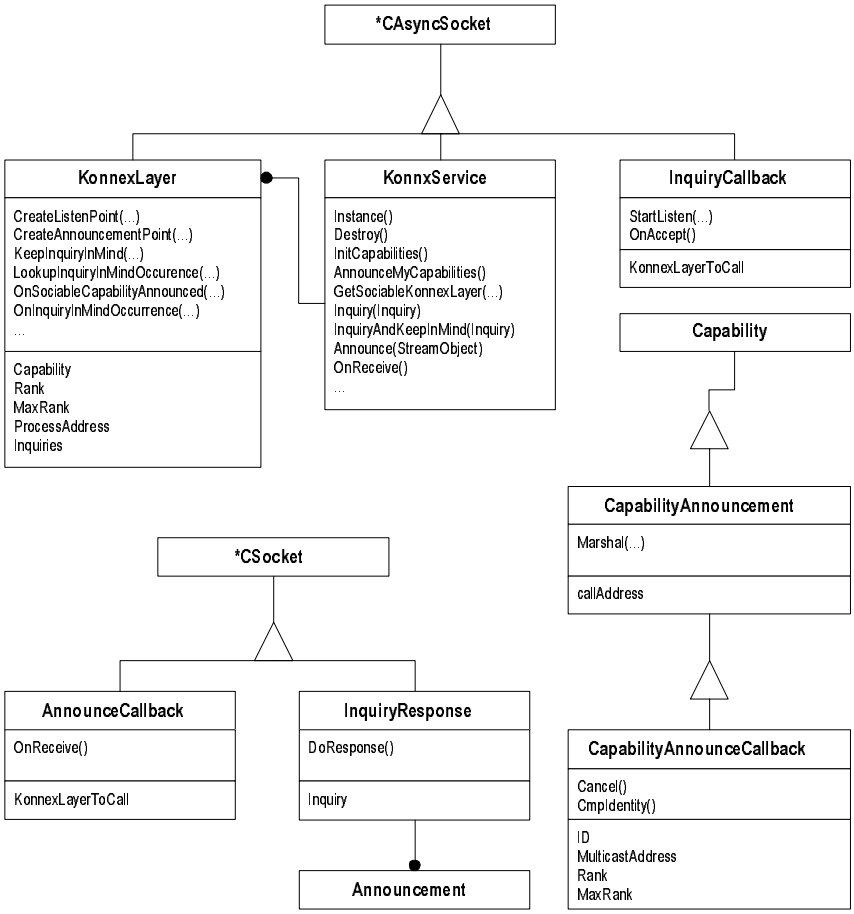

Abbildung 7.5 Objektmodell des Konnexionsdienstes

Komponentenintegration

Zunächst werden durch den Aufruf der Methode *InitCapabilities()* alle den aktuellen Fähigkeiten entsprechenden *KonnexLayer*-Instanzen erzeugt. Als Datenbasis dienen die Informationen des *ClassBrokers*. Mit Hilfe der Methode *CreateListenPoint()* werden dann alle *KonnexLayer*-Instanzen als Server konfiguriert. Der *KonnexService* meldet sich im Anschluß bei der DMSMS-Multicast-Basisgruppe an und zwar für jede im Rechner installierte Netzwerkkarte. Dann wird für jeden *KonnexLayer* eine *Capability-Announcement*-Instanz erzeugt; sie dient lediglich dazu, die Fähigkeiten durch Aufruf der Methode *Marshal(...)* in sendefähig Pakete zu überführen. Das so erzeugte Datenpaket wird in der DMSMS-Multicast-Basisgruppe veröffentlicht.

Bei allen Komponenten wird über die Methode *OnReceive()* das Eintreffen des veröffentlichten Paketes signalisiert. Pro entgegengenommenem Paket wird ein *CapabilityAnouncement*-Objekt erzeugt, dessen Attribute mittels der Methode *Marshal(...)* mit exakt denselben Werten besetzt werden, wie sie der Sender zuvor definiert hatte. Mit

diesen Informationen kann nun über die Methode *GetSociableKonnexLayer(Capability)* geprüft werden ob eine verträglicher *KonnexLayer*-Instanz existiert. Ist das der Fall, wird der *KonnexLayer*-Instanz über die Methode *OnSociableCapabilityAnnounced()* die Existenz der verträglichen Komponente mitgeteilt.

Nun prüft jede betroffene *KonnexLayer*-Instanz, ob sie selbst bereits im DMSM-System integriert ist. Wenn nein, initialisiert sie sich zunächst selbst, ansonsten berechnet sie den neuen maximalen Rang. Anschließend wird ein *CapabilityAnnouncement-Callback*-Objekt erzeugt. Dieses Objekt hat die Aufgabe, nach der errechneten Zeitverzögerung T_D (siehe Abbildung 5.18) einen Rückruf zu starten. Wird vor Ablauf der Zeit von der Verursacherkomponente ein Stopsignal veröffentlicht, können alle *KonnexLayer*-Instanzen die Rückrufaktion mittels der Methode *Cancel()* des Rückrufobjektes abbrechen. Wenn die Zeit aber abläuft, aktiviert sich das Objekt selbst und versucht eine Verbindung zu der *Konnexlayer*-Instanz der Verursacherkomponente aufzubauen. Die Verbindung kommt zustande, solange die angerufene *KonnexLayer*-Instanz als Server konfiguriert ist. Ansonsten wird der Ruf abgewiesen. Kommt eine Verbindung zustande, übermittelt das Rückrufobjekt die Daten der eigenen *KonnexLayer*-Instanz. Unabhängig davon, ob eine Verbindung zustande kam oder nicht, gibt sich das Rückrufobjekt im Anschluß sofort selbst auf.

Sobald sich eine Komponente auf der Serveradresse zurückgemeldet hat, ist die Integration der Komponente abgeschlossen. Sie konfiguriert die betroffene *KonnexLayer*-Instanz durch Aufruf der Methode *CreateAnnouncementPoint(...)* für das Veröffentlichen und Empfangen von Anfragen und Angeboten. Der Servermodus ist damit abgeschaltet. *CreateAnnouncementPoint(...)* meldet die KonnexLayer-Instanz bei der entsprechenden Multicastgruppe an.

Anfrage- und Angebotsbehandlung

Angebote können mittels der Methode *Inquiry(...)* veröffentlicht werden. Dazu instanziiert die Anwendung ein *Inquiry*-Objekt und füllt die Filterbedingungen nach Belieben aus. Anschließend übergibt sie das Objekt dem Konnexionsdienst. Dieser leitet es an jeden Konnexionslayer weiter. Jeder Konnexionslayer erzeugt ein *InquiryCallback*-Objekt und ruft dessen Methode *StartListen(...)* auf. *StartListen(...)* installiert einen Server und veröffentlicht die Anfrage inklusive der Serveradresse innerhalb seiner Multicastgruppe. Alle *KonnexLayer*-Instanzen innerhalb der Gruppe nehmen die Anfrage an und sammeln alle lokalen Stream-Objekte, die der Anfrage entsprechen. Aus dieser Sammlung wird eine *Announcements*-Instanz gebildet, die, sobald sie mehr als ein Angebot enthält, als Grundlage für eine *InquiryResponse*-Instanz dient. Die *InquiryResponse*-Instanz meldet sich beim zuvor installierten Server der anfragenden *KonnexLayer*-Instanz. Sofern dieser noch installiert ist, instanziiert er ein *AnnounceCallback*-Objekt, das die Angebote des *InquiryResponse*-Objektes annimmt.

Diese etwas kompliziert anmutende Implementierung ist für die Umsetzung der inhärenten Parallelitäten, wie sie in Kapitel 6.2 vorgeschlagen wurden, notwendig.

7.1.3 Der Transportservice *TransportService*

Die Schnittstelle des Transportdienstes präsentiert sich der Applikation nur mit der einzigen Methode *InitiateTransport(...)*. Die Adresse der Schnittstelle erhält die Applikation über die statische Methode *TransportService::Instance()*. Die Adresse ist während der gesamten Laufzeit der Applikation gültig. Erst mit Aufruf der Methode *Destroy()* wird der Dienst beendet und die Adresse ungültig. Beim ersten Aufruf dieser Methode wird der Transportdienst initialisiert.

Während der Initialisierung wird der *ClassBroker* angewiesen, alle Informationen der aktuell verfügbaren dynamischen Objekte zu sammeln und die *StreamObjektTemplate*-Liste auszufüllen. Danach wird eine *ObjectBroker*-Instanz erzeugt. Zum Abschluß der Initialisierung wird ein Server installiert, bei dem sich Komponenten anmelden müssen, wenn sie einen Datentransport einleiten wollen.

Als *Observer* beobachtet der Transportdienst den *ClassBroker*, um die Integration bzw. das Entfernen dynamischer Objekte zu erfahren. Als *Subject* kann der Transportdienst von der Applikation beobachtet werden, um ggf. ebenfalls über Änderungen der Fähigkeiten informiert zu werden.

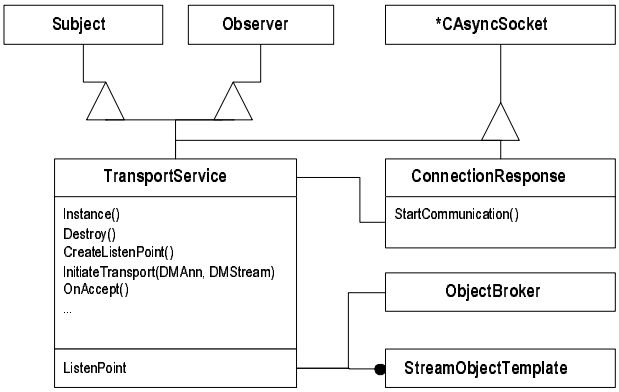

Abbildung 7.6 Objektmodell des Transportdienstes

Die Anwendung kann einen Datentransport einleiten, indem sie die Methode *InitiateTransport(DMAnnouncement, DMStreamObject)* aufruft. Die Instanzen der Klassen *DMAnnouncement* und *DMStreamObject* kann die Anwendung z.B. dem *StreamManager*- bzw. dem *AnnouncementMananger*-Objekt des Registraturdienstes entnehmen.

Die Methode *InitiateTransport(...)* implementiert exakt den in Kapitel 5.3.3 Abbildung 5.20 definierten Algorithmus. Da er innerhalb der Methode sequentiell abgearbeitet wird, ist er in der Implementierung einfach nachzuvollziehen und soll deshalb nicht weiter beschreiben werden.

Der Aufbau der Kommunikationsverbindungen zu den Gegenstellen wird durch das Anmelden beim Server des jeweiligen Transportdienstes eingeleitet. Die Adressen können den Angeboten entnommen werden. Der Server der angerufenen Transportdienste

erzeugt für jede Verbindung ein *ConnectionResponse*-Objekt. Diese Objekte arbeiten den im Kapitel 0 Abbildung 5.21 definierten Algorithmus ab.

Die Kommunikationsverbindungen zu den einzelnen Transportdiensten werden nicht abgebrochen. Sie werden an die *DMSRCPComm*-Objekte der am Datentransport beteiligten Stream-Engine-Instanzen weitergereicht. Diese nutzen die bestehenden Verbindungen für die Realisierung des virtuellen Steuerungsbusses. Damit wird ein weiterer Verbindungsaufbau zwischen den Steuerungsobjekte überflüssig.

7.1.4 Der Registraturdienst *RegistrationService*

Die Schnittstelle des Registraturdienstes bietet ausschließlich Methoden, die indirekt den internen Stream- bzw. Angebotsmanager bedienen. Die Adresse der Schnittstelle erhält die Applikation über die statische Methode *RegistrationService::Instance()*. Die Adresse ist während der gesamten Laufzeit der Applikation gültig. Erst mit Aufruf der Methode *Destroy()* wird der Dienst beendet und die Adresse ungültig. Beim ersten Aufruf dieser Methode wird der Registraturdienst initialisiert.

Während der Initialisierung werden lediglich die beiden Klassen *DMStreamManager* und *DMAnnouncementManager* instanziiert. Der beiden Manager enthalten direkt nach der Initialisierung keine Stream-Objekte bzw. Angebote.

Als *FileWatchDog* kann der Registraturdienst jede Änderung bzgl. lokaler Stream-Objekte mit expliziter Assoziation beobachten und ggf. entsprechend handeln. Als *Subject* kann der Registraturdienst beobachtet werden. Er signalisiert jedem Beobachter das Eintreffen von Angeboten und die Integration bzw. das Entfernen lokaler Stream-Objekte.

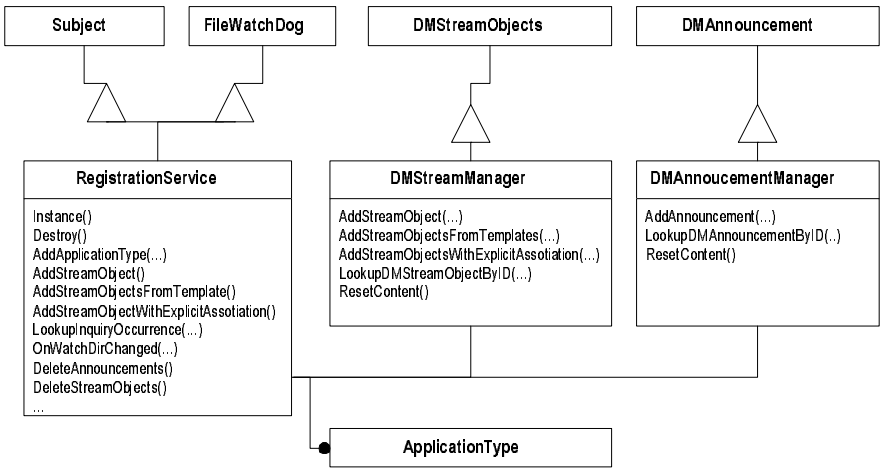

Abbildung 7.7 Objektmodell des Registraturdienstes

Der Registraturdienst muß nach seiner Initialisierung von der Anwendung konfiguriert werden. Als erstes meldet sich die Applikation über die Methode *AddApplikationType(...)* an. Eine Applikation kann diese Methode mehrmals aufrufen, wenn sie mehrere Funktionalitäten implementiert.

Sobald die Applikation sich angemeldet hat, sollte sie die *DMStreamManager*-Instanz anweisen, die Liste der lokalen Stream-Objekte zu füllen. Zunächst können mit Hilfe der Methode *AddStreamObjectsFromTemplates()* alle Stream-Objekte mit impliziter Assoziation eingefügt werden. Anschließend können durch Aufruf der Methode *AddStreamObjectsWithExplicitAssociation(...)* die Stream-Objekte mit expliziter Assoziation eingefügt werden.

Die Stream-Objekte mit impliziter Assoziation werden direkt aus den Angaben in den *StreamObjectTemplate*-Instanzen des Transportdienstes abgeleitet. Stream-Objekte mit expliziter Assoziation werden nach dem in Kapitel 5.4.3 vorgeschlagenen Ablauf integriert. Die verallgemeinerte Bezeichnung „Ortsangabe“ aus Kapitel 5.4.3 ist im konkreten Fall der Beispielimplementierung eine Pfadangabe, die der Methode *AddStreamObjectsWithExplicitAssociation(...)* übergeben wird.

7.2 Demonstratorsystem

Mit Hilfe einer Anwendung, einem Transportobjekt und mehreren Stream-Engine-Objekten soll die Beispielimplementierung des DMSM-Systems genutzt werden. Ziel ist es dabei, alle in der Beispielimplementierung realisierten Funktionalitäten des DMSM-Systems testen und vorführen zu können.

Abhängig von den installierten Stream-Engine-Objekten lassen sich Audio-Encoder, Audio-Decoder und Audio-Server-Komponenten für Aufnahme und Wiedergabe konfigurieren und über die Anwendung nahezu beliebig zusammenschalten. Eine Beispieltopologie des Demonstratorsystems stellt Abbildung 7.8 dar. Mit dieser Konfiguration lassen sich leicht Audio-On-Demand-Szenarien (Live oder Konserve) realisieren. Das Einbringen eines neuen Streams in das System gestaltet sich ebenso einfach wie der Abruf eines Streams.

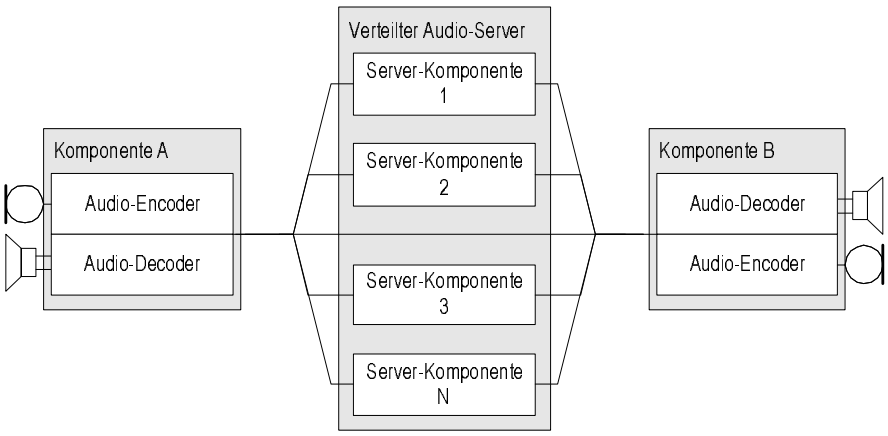

Abbildung 7.8 Beispielkonfiguration des Demonstratorsystems

Der Aufwand für die Beispielimplementierung soll durch folgende Maßnahmen beschränkt werden:

- Applikation, Stream-Engine-Objekte und Transportobjekte werden in demselben Adreßraum betrieben.
- Für den virtuellen Steuerungsbus wird kein Multicast-Protokoll eingesetzt.
- Der verteilte Server darf während einer Aufzeichnung nur aus einer geraden Komponentenzahl bestehen.
- Als Transportprotokoll für Stream-Objektdaten soll nur TCP eingesetzt werden.
- Ressourcen werden nach dem Best-Effort-Prinzip vergeben.
- Bei der Existenz von Repliken wird ein beliebiges gewählt.

7.2.1 Einfaches TCP-Transport-Objekt

Das TCP-Transportobjekt implementiert das Wissen über den Verbindungsaufbau einer TCP-Verbindung.

Die Regel für den Verbindungsaufbau lautet: Eine Gegenstelle muß vor dem Verbindungsaufbau ihre Bereitschaft signalisieren. Dies geschieht durch Binden eines Listen-Socket. Anschließend kann der Initiator über das Connect-Kommando den Verbindungswunsch äußern, den die Gegenstelle über das Accept-Kommando annehmen kann.

Die Methode *TransportTo(Address, SizeOfAddress, DataDirection, IAmInitiator)* implementiert genau diese Regel. Abhängig vom Argument *IAmInitiator* richtet die *SimpleTCP*-Instanz entweder ein Listen-Socket ein oder sie führt das Connect-Kommando mit der Zieladresse *Address* aus. Die Adresse hat der Transportdienst zuvor von der Gegenstelle erhalten. Die Gegenstelle wiederum hat die Adresse über die Methode *HowToCallMe(&Address, &SizeOfAddress)* ermittelt. Das Argument *DataDirection* ist für den Aufbau einer TCP-Verbindung irrelevant.

Die Methode *GetConnectionPoint()* liefert einen Socket-Bezeichner zurück, der als Argument für Schreib- und Lesebefehle benutzt werden kann.

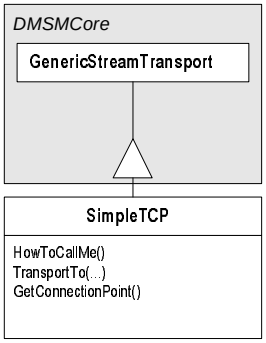

Abbildung 7.9 Objektmodell des TCP-Transportobjektes

7.2.2 Einfache Stream-Engine-Objekte

Die Stream-Engine-Objekte implementieren das Wissen über den Datentransport. Der Datentransport findet immer statt, entweder zwischen einer Quelle und einer oder mehreren Senken oder zwischen einer Senke und einer oder mehreren Quellen.

DIPCM-Stream-Format

Mit Hilfe des Demonstratorsystems sollen auch Szenarien mit verteilten Stream-Objekten ermöglicht werden. Deshalb soll das System quasi-parallel verteilte Streams verarbeiten können.

Die Verarbeitung quasi parallel verteilter Streams erfordert ein spezielles Datenformat, das Aufschluß über die innere Struktur eines Streams gibt. Für das Demonstratorsystem wurde dazu das sogenannte DIPCM-Format entworfen. DIPCM steht für Distributed-PCM-Format und erweitert die Standard-PCM-Daten um Synchronisations- und Strukturdaten. Jedem DIPCM-Datenblock wird ein *DIPCMHEADER* vorangestellt.

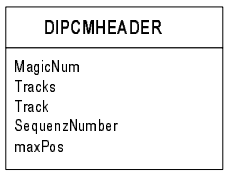

Abbildung 7.10 Header des DIPCM-Stream-Formates

Die Attribute des Headers geben Auskunft über die gesamte Anzahl der Spuren, die Spurnummer und die Sequenznummer des anschließenden DIPCM-Datenblocks und die maximale Position. Die maximale Position ist normalerweise konstant und gilt als Grenzwert für den Steuerbefehl *SetPos()* eines *GenericStreamControl*-Objektes; falls ein Stream-Objekt im Laufe der Zeit wächst, steigt der Wert *maxPos* monoton an.

Die Länge eines DIPCM-Blocks richtet sich nach dem Wert des Attributes *Tracks*. Der Inhalt eines DIPCM-Blocks besteht aus jedem N-ten (N=*Tracks*) Sample eines PCM-Blocks der dem PCM-Datenstrom entnommen wurde. Die Spurnummer *Track* bestimmt dabei den Versatz von *v* (*v=Track*) Samples bzgl. der anderen *N-1* Teilblöcke.

Es wird angenommen, daß die Länge des ursprünglichen PCM-Blocks konstant bzgl. der gesamten Übertragungszeit und beiden Endpunkten der Übertragung bekannt ist.

Der Einfachheit halber wird festgelegt, daß die Länge des ursprünglichen PCM-Blocks ein Vielfaches des Teilungsverhältnisses 1/N mit N=*Tracks* ist. Daraus ergibt sich die gleiche Blockgröße für alle N Teil-Blöcke. Außerdem werden nur gerade Teilungsverhältnisse zugelassen.

DIPCM-Encoderobjekt

Das DIPCM-Encoderobjekt hat die Aufgabe, während der Instanziierung das Audio-Device eines Rechners so zu konfigurieren, daß Audiosignale, die am Analogeingang der Audio-Hardware anliegen, als digitaler Datenstrom zur Verfügung stehen.

Der aktive Prozeß, der für den Datentransport verantwortlich ist, leitet dann den Datenstrom über die Verbindungsendpunkte zu den Gegenstellen. Die Verbindungsendpunkte werden über die Methode *SetEngineOutput(ConnectionPoints)* angegebenen.

Abhängig von der Anzahl der Verbindungsendpunkte wird innerhalb der Methode *ThreadAndSendBlock(...)* der *TCPSink*-Instanz ein PCM-Datenblock der *PCMSource*-Instanz in DIPCM-Blöcke zerteilt und an alle Gegenstellen gesendet.

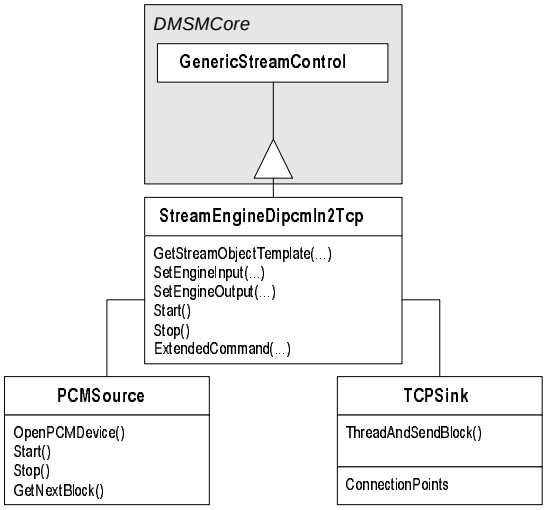

Abbildung 7.11 Objektmodell des DIPCM-Encoderobjektes

DIPCM-Decoderobjekt

Das DIPCM-Decoderobjekt hat die Aufgabe, während der Instanziierung das Audio-Device eines Rechner so zu konfigurieren, daß ein digitaler Datenstrom in analoge Audiosignale umgewandelt wird und am Analogausgang der Audio-Hardware zur Verfügung steht.

Der aktive Prozeß des Stream-Engine-Objektes leitet den Datenstrom, der über die Verbindungsendpunkte eintrifft, an die *PCMSink*-Instanz weiter. Die Verbindungsendpunkte werden durch die Methode *SetEngineInput(ConnectionPoints)* festgelegt. Die *TCPSource*-Instanz ist für die Zusammensetzung des Gesamtdatenstroms aus den Teil-Datenströmen verantwortlich.

Die Methode *ExtendedCommand(...)* stellt einer Applikation die Möglichkeit bereit, ein adaptives IIR-Filter[20] in den Gesamtdatenstrom einzuschleifen. Damit sollen bei späteren Tests Störungen beseitigt werden können, die durch das Fehlen von Teildatenströmen verursacht werden. Außerdem wurde eine frequenzmodelierende Geschwindigkeitsregelung implementiert, die es erfordert, daß die Grenzfrequenz des Ausgangssignals an die geänderte Samplingfrequenz anpaßt wird.

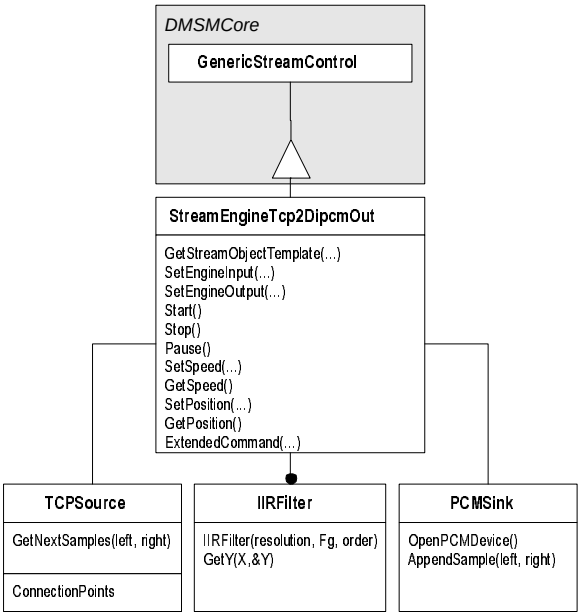

Abbildung 7.12 Objektmodell des DIPCM-Decoderobjektes

DIPCM und WAV-Dateiwiedergabeobjekte

Die Dateiwiedergabeobjekte haben die Aufgabe, während ihrer Instanziierung eine Datei zu öffnen und die in ihr gespeicherten digitalen Audiodaten als Datenstrom zur Verfügung zu stellen.

Der aktive Prozeß, der für den Datentransport verantwortlich ist, leitet dann den Datenstrom über die Verbindungsendpunkte zu den Gegenstellen. Die Verbindungsendpunkte werden durch die Methode *SetEngineOutput(ConnectionPoints)* festgelegt.

Die Assoziation eines Engine-Objektes mit einer Datei wird durch die Metadaten des Stream-Objektes ermöglicht, das der Methode *SetEngineInput(StreamObjekt)* übergebenen wird.

Abhängig von der Anzahl der Verbindungsendpunkte wird innerhalb der Methode *ThreadAndSendBlock(...)* der *TCPSink*-Instanz ein PCM-Datenblock der *WavSource*- bzw. *DIPCMSource*-Instanz in DIPCM-Blöcke zerteilt und an alle Gegenstellen gesendet. Falls weniger Gegenstellen angeschlossen sind, als tatsächliche Spuren vorhanden,

[20] Das IIR-Filter wurde mit den in [Tie89] vorgeschlagenen Regeln entworfen. Aufgrund der kaum möglichen Realisierung adaptiver IIR-Filter wurde die Adaptivität durch gemultiplexte Filterbänke erreicht.

werden nur alle anwesenden Gegenstellen mit Daten versorgt. Die Daten, die eigentlich für die fehlenden Gegenstellen bestimmt sind, werden nicht versendet.

Die Dateiwiedergabeobjekte können bei einer 1:M-Beziehung in zwei Rollen eingesetzt werden, zum einen als Wurzel-Instanz, zum anderen als Blatt-Instanz. Zusammen mit dem DIPCM-Aufzeichnungsobjekt ergibt sich damit die Möglichkeit, parallel verteilte Streams in konzentrierte zu überführen und umgekehrt.

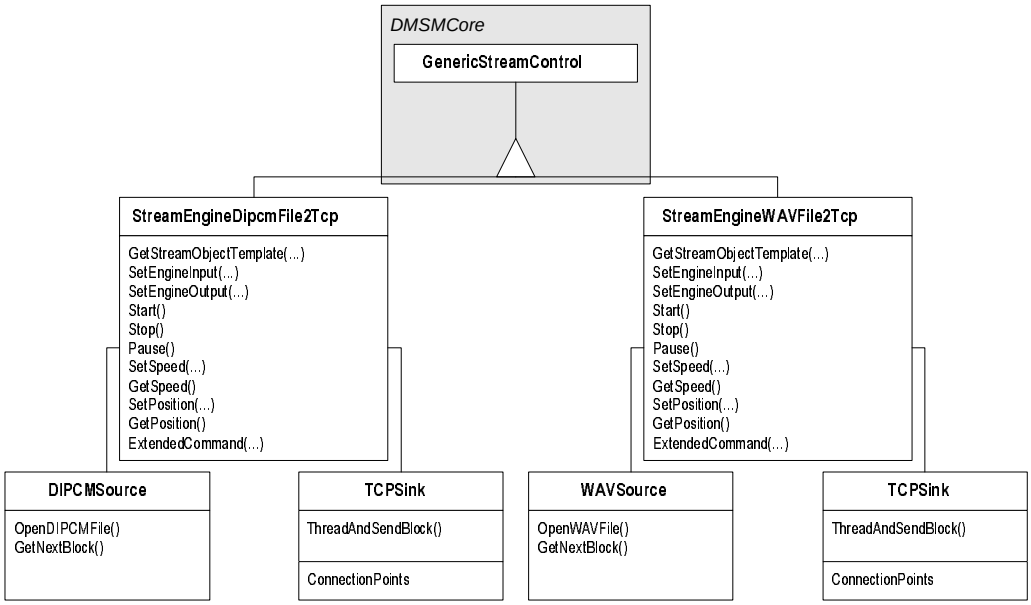

Abbildung 7.13 Objektmodell des DIPCM- und WAV-Wiedergabeobjektes

DIPCM-Aufzeichnungsobjekt

Das DIPCM-Aufzeichnungsobjekt hat die Aufgabe, während der Instanziierung eine Datei zu öffnen, um in ihr einen Datenstrom digitaler Audiodaten zu speichern.

Der aktive Prozeß des Stream-Engine-Objektes leitet dann den Datenstrom, der über die Verbindungsendpunkte eintrifft, an die *DIPCMSink*-Instanz weiter. Die Verbindungsendpunkte werden mit Hilfe der Methode *SetEngineInput(ConnectionPoints)* angegebenen.

Die Assoziation eines Engine-Objektes mit einer Datei wird durch die Metadaten des Stream-Objektes ermöglicht, das der Methode *SetEngineOutput(StreamObjekt)* übergebenen wird.

Das Aufzeichnungsobjekt kann in zwei Rollen eingesetzt werden, zum einen als Wurzel-Instanz, zum anderen als Blatt-Instanz bei einer 1:M-Beziehung. In der Rolle einer Wurzel-Instanz muß die *TCPSource*-Instanz einen Gesamtdatenstrom aus den Datenströmen der M Gegenstellen bilden, bevor eine Übergabe der Daten an die *DIPCMSink*-Instanz erfolgen kann. Falls die tatsächliche Anzahl der aktiven Gegenstellen kleiner ist

als die notwendige, werden die fehlenden Daten aus den tatsächlich angelieferten Daten interpoliert.

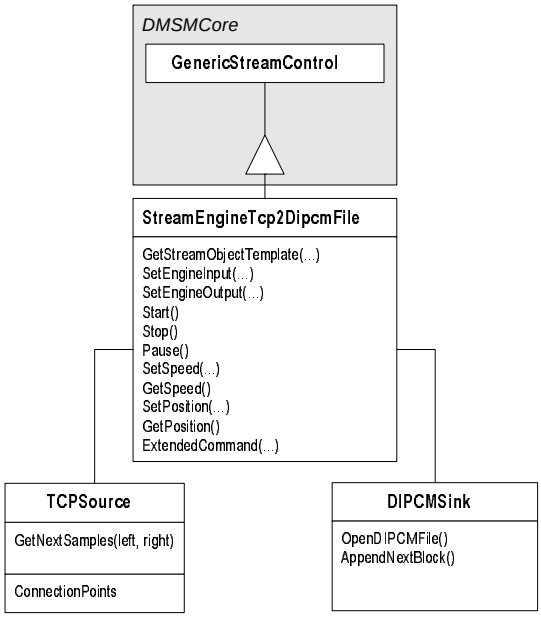

Abbildung 7.14 Objektmodell des DIPCM-Aufzeichnungsobjektes

7.2.3 DMSMS-Testapplikation

Die DMSMS-Testapplikation implementiert lediglich ein User-Interface zur Nutzung der DMSM-Dienste. Nachdem sich die Applikation die Adresse der Schnittstellen aller Dienste mit Hilfe der jeweiligen *Instance()* Methoden geholt hat, meldet sie sich als Client und Server beim Registraturdienst an. Anschließend weist sie den Registraturdienst an, die lokalen Stream-Objekte durch den Stream-Manager aufnehmen zu lassen. Dabei spezifiziert die Applikation auch die Verzeichnisse, in denen Daten explizit assoziierter Stream-Objekte abgelegt sind und abgelegt werden sollen. Der Inhalt des Stream-Managers wird dann in der Liste *Local Stream Objekts* angezeigt.

Die Funktionalität einer Komponente wird in der Liste *Application Types*, die Fähigkeiten in der Liste *Capabilities* dargestellt.

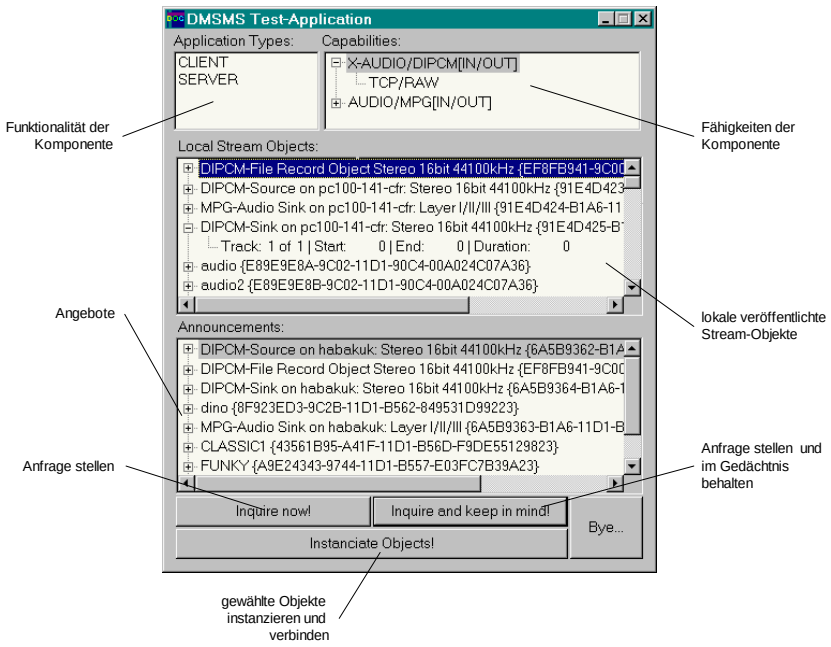

Abbildung 7.15 Userinterface der DMSMS-Testapplikation

Bei Betätigung des Knopfes *Inquirie now* wird eine Instanz der *Inquiry*-Klasse erzeugt und leer dem Konnexionsdienst übergeben. Damit wird eine allgemeine Anfrage in allen Konnexionslayern gestellt, so daß alle verträglichen Komponenten ihre lokalen Stream-Objekte als Angebote der anfragenden Instanz offerieren. Die eingehenden Angebote werden in der Liste *Announcements* dargestellt. Durch Betätigen des Knopfes *Inquiry and keep in mind* wird ebenfalls eine allgemeine Anfrage gestellt, mit dem Unterschied, daß anschließend die Anfrage gespeichert wird, so daß Angebote, die im Anschluß an die gestellte Anfrage veröffentlicht werden, ebenfalls der Anfrage zugeordnet werden können.

Enthält jede der beiden Listen *Local Stream Objects* und *Announcements* mindestens einen Eintrag, kann in jeder Liste ein Eintrag gewählt werden. Beide markierten Objekte

können dann durch Drücken des Knopfes *Instanciate Objects* instanziiert und verbunden werden. Dazu werden die gewählten Objekte dem Transportdienst über den Methodenaufruf *InitiateTransport*(*Announcement*, *LocalStreamObject*) mitgeteilt.

Stehen beide markierten Objekte in einer Komplementärbeziehung und tritt kein Fehlerzustand ein, kommt eine Datenübertragung zustande. Die Applikation erhält dann eine gültige Adresse der Steuerungsschnittstelle der am Transport beteiligten Stream-Engine. Die Schnittstelle wirkt über das Steuerungsobjekt indirekt auf den virtuellen Steuerungsbus, so daß jede an der Datenübertragung beteiligte Instanz durch die Bedienung der Schnittstelle gesteuert wird.

Da nur *eine* Applikation für das Demonstratorsystem implementiert werden soll, muß die Benutzerschnittstelle Bedienungselemente für alle benötigten Funktionalitäten der Steuerungsschnittstelle bereithalten. Sobald eine Steuerungsschnittstelle eine Funktionalität nicht implementiert, wird das entsprechende Bedienungselement insensitiv konfiguriert.

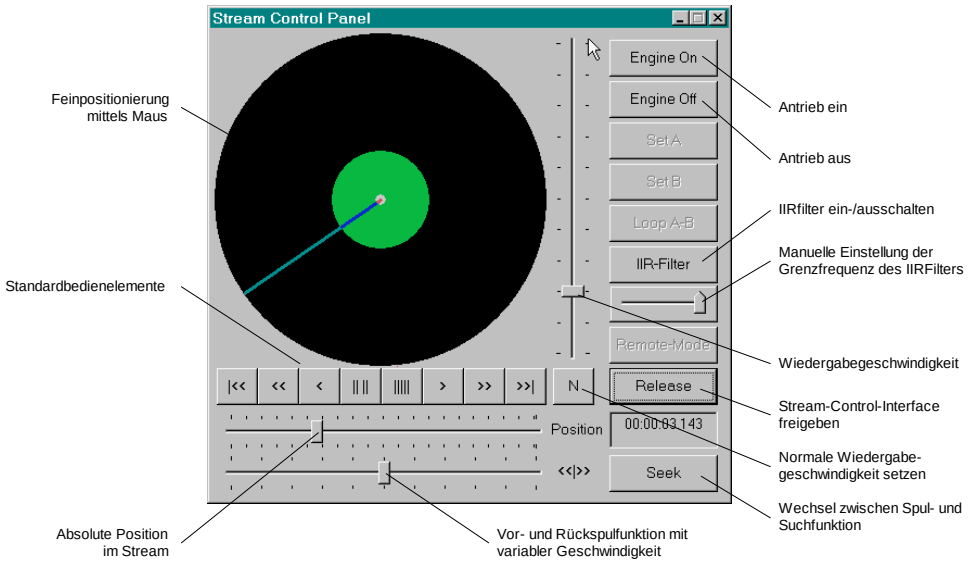

Abbildung 7.16 Userinterface des Stream-Control-Panels

8 Bewertung

Alle Ergebnisse und ihre Bewertung sollten als Basis für einen weiteren Iterationsschritt dienen, der im Rahmen dieser Arbeit nicht durchgeführt werden kann. Sollte das Ziel verfolgt werden, das DMSM-System zur Produktreife zu bringen, müssen in jedem Fall weitere Durchläufe von Analyse, Entwurf und Evaluation folgen.

Als Bewertungsgrundlage dient die Beispielimplementierung des DMSM-Systems, das Demonstratorsystem und die damit realisierten Testszenarien. Alle Tests wurden innerhalb des Intranets der SICAN GmbH durchgeführt.

8.1 Erweiterbarkeit des Systems

Die Implementierung des Demonstratorsystems ist bereits als Testszenario anzusehen. Es wurde der Nachweis erbracht, daß ein Anwendungsprogrammierer nur wenig Kenntnisse über das DMSM-System besitzen muß, um komplexe On-Demand-Szenarien zu realisieren.

Ein Entwickler von Stream-Engine-Objekten bzw. Transportobjekten muß sich nur in die Schnittstellen zum DMSM-System einarbeiten; er muß keine genauen Kenntnisse über den Aufbau des DMSM-Systems besitzen. Für den Nachweis der einfachen Erweiterbarkeit sollte das System um die Fähigkeit erweitert werden, MPEG-Audiostreams zu transportieren. Dazu wurden lediglich zwei Stream-Engine-Klassen *StreamEngineMpg2Tcp* und *StreamEngineTcp2MpgOut* implementiert und als DLL-Module in das Verzeichnis der Applikation abgelegt. Als Transportklasse sollte die *SimpleTCP*-Klasse benutzt werden. Das System war ohne weitere Änderungen sofort in der Lage, MPEG-Audiostreams der Layer I/II/III zu verarbeiten.

Die herausragende Eigenschaft, das System sehr schnell um neue Formate bzw. Netztechnologien zu erweitern, wird noch dadurch unterstrichen, daß bei der Erweiterung ei-

ner Komponente um neue Transport- bzw. Stream-Engine-Objekte der laufende Betrieb einer Applikation nicht unterbrochen werden muß.

Neben der Formaterweiterung des Systems um MPEG-Audio wurde während der Testphase ein weiters Transportobjekt implementiert, um Stream-Objektdaten über ATM/AAL5 zu versenden. Es zeigte sich, daß die Wahl einer allgemeinen Verbindungsmethode die Wiederverwendung von Stream-Engine-Objekten in verschiedenen Netztechnologien möglich macht, aber auf die Ausnutzung spezifischer Kanaleigenschaften zum Zwecke der Optimierung verzichtet werden muß.

8.2 Komponentenintegration, Anfragen und Angebote

Vor jedem Datentransport muß eine Applikation sich zunächst in das DMSM-System integrieren, und dann über öffentliche Anfragen ihr Interesse äußern. Erst wenn Angebote empfangen wurden, kann ein Datentransport erfolgen. Die bis dahin notwendigen Vorgänge innerhalb des DMSM-Systems müssen für einen reibungslosen Betrieb schnell und zuverlässig durchgeführt werden.

Für die Komponentenintegration wurde der sogenannte Komponentenintegrations-Algorithmus entwickelt und implementiert. Während der Testphase bestätigte sich das erwartete Verhalten des Algorithmus – mit zunehmender Anzahl an verträglichen Komponenten wird es immer wahrscheinlicher, daß die Integration ohne Verzug erfolgt. Außerdem war es nicht möglich, eine gescheiterte Komponentenintegration nachzuweisen. Bei Komponentenzahlen von weniger als 5 machte sich lediglich die verzögerte Integration bemerkbar, nachdem Komponenten mehrere Male ohne korrekte Abmeldung beendet wurden. Da solche Verzögerungen im Sekundenbereich liegen und nur während des Starts der Applikation auftreten, fallen sie nicht unangenehm auf.

Um die Belastung einer anfragenden Komponente zu maximieren, werden Anfragen im Demonstratorsystem immer allgemein gestellt, so daß alle Stream-Objekte aller verträglichen Komponenten der anfragenden Instanz angeboten werden. Durch die parallele Verarbeitung der eingehenden Angebote wird die Zeit, die das Fenstersystem für das Füllen der Listen benötigt, in den meisten Fällen größer als das Sammeln aller Angebote im Angebotsmanager sein. Der Anwender wird deshalb keinen Unterschied zwischen der Anfrage an eine zentrale Datenbank und der Anfrage innerhalb des DMSM-System feststellen können. Die Zahl der im Netz verteilten Stream-Objekte betrug während der Testphase ca. 1000. Man könnte überlegen, ob die Anzahl der anzunehmenden Angebote begrenzt werden sollte, denn ein Benutzer wird vermutlich bei mehr als 1000 Angeboten eher seine Anfrage spezifischer formulieren als alle 1000 Angebote zu sichten.

8.3 Stream-Transport und Stream-Steuerung

Wenn eine Komponente ein Stream-Objekt anbietet, impliziert dies, daß sie mit dem Transport des Streams einverstanden ist. Es bedarf also keiner weiteren Zustimmung der am Transport beteiligten Komponenten, so daß der Stream-Transport ohne Verzug

eingeleitet wird, sobald die Applikation den Transportdienst damit beauftragt. Wie erwartet, ist die Dauer des Verbindungsaufbaus vernachlässigbar klein.

Sobald ein Datentransport initiiert worden ist, kann über die Schnittstelle des Stream-Engine-Objektes der Datentransport gesteuert werden. Um die Leistungsfähigkeit der Steuerung bewerten zu können, wurde eine Folgeregelung realisiert. Das Stream-Control-Panel (siehe Abbildung 7.16) simuliert dazu eine rotierende Scheibe mit einer bestimmten Masse *m*, die über ein Drehmoment *M* angetrieben wird. Das eingeprägte Drehmoment läßt sich mit Hilfe der Maus direkt über die rotierenden Scheibe des Stream-Control-Panels modulieren oder indirekt über die Geschwindigkeitsregler. Die resultierende Winkelgeschwindigkeit wird an die Stream-Engine übermittelt. Die damit erzielbaren Effekte gleichen denen, die mit einem Analogplattenspieler produziert werden können. Es zeigte sich, daß die Qualität der erzielbaren Effekte direkt von der Leistungsfähigkeit des virtuellen Steuerungsbusses abhängt. Da bei der Beispielimplementierung der virtuelle Steuerungsbus durch TCP-Verbindungen realisiert wurde, wirkt sich die Netzauslastung negativ auf die Leistungsfähigkeit der Steuerung aus. Bei geringer Netzauslastung ist unabhängig vom Einspeispunkt der Steuerungssignale die Phasenverschiebung der Folgeregelung vernachlässigbar klein, während sie bei zunehmender Netzlast im Mittel steigt und starken Schwankungen unterworfen ist. Für die Echtzeitfähigkeit des Steuerbusses ist es deshalb erforderlich, daß das Netzwerk, über das der virtuelle Steuerungsbus realisiert wird, echtzeitfähig ist.

Ein weiterer Nachteil der Beispielimplementierung des Steuerungsbusses ist, daß die Leistungsfähigkeit von der Anzahl der zu steuernden Komponenten abhängt – einerseits bedingt durch die begrenzte Anzahl der TCP-Verbindungen pro Komponente und andererseits durch den zeitlichen Versatz der Steuerungskommandos. Eine unscharfe Grenze der Spurzahl eines quasi parallel verteilten Streams wurde beim Demonstratorsystem bei ca. 32 festgestellt. Die Grenze ist von der Netzauslastung abhängig. Die eingeschränkte Leistungsfähigkeit des Steuerungsbusses könnte durch Verwendung eines Multipoint-Protokolls aufgehoben werden.

Bei der Erzeugung bzw. beim Abruf quasi parallel verteilter Streams zeigte sich, daß die Auslastung der Komponente des Initiators unabhängig von der Spurzahl ist. Zwischen der Beanspruchung jeglicher Ressourcen auf den Gegenstellen und der Spurzahl wurde ein umgekehrt proportionaler Zusammenhang festgestellt. Dieses Verhalten entspricht exakt den Erwartungen.

8.4 Weiterführende Szenarien

Neben den klassischen On-Demand-Szenarien sind mit Hilfe des Demonstratorsystems auch weiterführende Szenarien realisierbar. Das Aufzeichnungsobjekt bietet die Möglichkeit, Stream-Objekte innerhalb des System zu produzieren, durch das Zusammenschalten von Stream-Engine-Objekten mit verschiedenen Medienformaten – aber gleicher Verbindungsmethode – lassen sich Konverter realisieren, und durch Einsatz geeigneter Audio-Hardware lassen sich Konferenzszenarien durchführen.

Gleichzeitiges Aufzeichnen und Wiedergeben

Bemerkenswert ist, daß die in Kapitel 5.2.7 geäußerte Auffassung – die Aufzeichnung ist ein Metazustand konservierter Daten – und ihre konsequente Berücksichtigung zu dem Ergebnis führten, daß bereits während einer Aufzeichnung ohne Einschränkung das Stream-Objekt abgerufen werden kann. Die einzige Schwierigkeit, die sich bei der Umsetzung ergab, war die Darstellung der kontinuierlich wachsenden Grenzen in der Benutzerschnittstelle.

Lokales multiplizieren des Objektdatenstroms

Bei der Durchführung von klassischen On-Demand-Szenarien ergibt sich die Anforderung, einen ausgegebenen Datenstrom lokal mitzuschneiden. Um diese Forderung zu erfüllen, müßte der Datenstrom auf verschiedenen Geräten gleichzeitig ausgeben werden können. Das Konzept der Stream-Engines sieht dies nicht explizit vor. Es wäre deshalb sinnvoll, das Konzept der Stream-Engines um eine interne virtuelle Busarchitektur zu erweitern. Es gäbe dann eine Root-Stream-Engine, die den Datentransport durchführt und steuert sowie die Daten auf einem virtuellen Datenbus bereitstellt. Eine weitere Klasse von Stream-Engines könnte sich dann auf den virtuellen Datenbus schalten und die Daten weiterverarbeiten.

Signalisierung und Einbinden externer Komponenten

Bei einem Konferenzszenario ist die Signalisierung ein wichtiger Aspekt. Innerhalb des DMSM-Systems kann sie beispielsweise einfach durch die Transportobjekte realisiert werden. Wenn aber auf das zusätzliche Kommunikationsmedium verzichtet werden soll – z.B. auf das Ethernet bei der Beispielimplementierung – oder eine Komponente außerhalb des DMSM-Systems plaziert werden soll, muß jede Komponente die Adressen der anderen bzw. externen Komponenten kennen. Ein klassisches Beispiel ist ein zu integrierendes ISDN-Telefon. Wollte man vom ISDN-Telefon aus eine DMSM-Komponente erreichen, muß der Anrufende die Adresse der Komponente wissen und der Angerufene muß durch die Instanziierung eines Transportobjektes die Erreichbarkeit realisieren. Am einfachsten könnte die Anbindung externer Komponenten über ein Gateway-Komponente erfolgen, damit die Adressen der extern angeschlossenen Geräte zentral administriert werden können.

9 Zusammenfassung

Aufgabe war es, ein System für das Management verteilter Streams zu entwerfen. Dabei sollte der gesamte Produktionsprozeß betrachtet werden, vom Erzeuger über den Anbieter bis zum Konsumenten. Das System sollte so konzipiert werden, daß es dezentral organisiert ist – es somit keine übergeordneten administrativen Instanzen gibt – und daß die mögliche verteilte Struktur eines Streams den Anwendern verborgen bleibt.

Die Kernprobleme, die es zu lösen galt, waren die dezentrale Verwaltung, die Irrelevanz der Datenflußrichtung[21] und die für den Anwender vollständig zu verdeckende innere Struktur verteilter Streams.

Zu Beginn erfolgte die detaillierte Problembeschreibung. Sie diente als Basis für die statische, dynamische und funktionale Modellierung.

Mit Hilfe der Objektmodellierung wurden die statischen Strukturen des DMSM-Systems erfaßt. Das Objektmodell zeigt Objekte im System, die Relationen zwischen den Objekten und die Attribute und Operationen, die jede Klasse von Objekten charakterisieren. Nach der Extraktion konkreter Klassen aus der Problembeschreibung wurden diese detailliert analysiert. Oft war es notwendig, dynamische wie funktionale Aspekte zu bedenken, um das Wesen der Objekte präzise erfassen zu können. Die wichtigsten Themen während der Objektmodellierung waren die Analyse der Stream-Objekte und ihrer inneren Struktur, die Ableitung wie auch die Analyse der DMSM-Dienste und die gundlegenden Konzepte zur Ressourcenverwaltung und Stream-Steuerung.

Während der dynamischen Modellierung wurden die Veränderungen von Objekten und ihren Relationen untersucht, die sich während der Laufzeit des Systems ergeben. Es wurden Konzepte entwickelt zur Regelung des Kontrollflusses, der Interaktionen und

[21] Die Irrelevanz der Datenflußrichtung resultiert aus der Forderung, den gesamten Produktionsprozeß abzudecken.

der Ausführungsreihenfolge von Operationen parallel zueinander aktiver Objekte innerhalb des DMSM-Systems.

Während der dynamischen Modellierung stand die dezentrale Organisation des Systems im Mittelpunkt. Das wichtigste Ergebnis ist, daß sich eine strukturlose Anhäufung von DMSM-Systemkomponenten eigenverantwortlich in eine höhere Ordnung begibt, indem jede Komponente ihre eigenen Fähigkeiten veröffentlicht und daraufhin ggf. von der Existenz einer Gruppe mit verträglichen Fähigkeiten erfährt. Die Tatsache, daß allein das Wissen einer Komponente über die Existenz einer verträglichen Gruppe sie darin integriert, ließ vermuten, daß das Verhalten des Systems unabhängig von der Komponentenzahl ist.

Die dezentrale Verarbeitung der verfügbaren Stream-Objekte wurde ebenfalls jeder einzelnen Komponente zugeteilt. Der Ansatz lautete, daß Stream-Objekte sich selbst präsentieren sollen und selbst über das prozedurale bzw. deklarative Spezialwissen für ihren eigenen Transport verfügen. Die Forderung nach der Irrelevanz der Datenflußrichtung und das Verbergen der inneren Stream-Struktur machten es notwendig, das dazu notwendige Spezialwissen außerhalb des Systems zu lagern.

Es wurde das Konzept der dynamischen Transportobjekte entwickelt. Spezialwissen über den Verbindungsaufbau und den Datentransport wird in konzentrierter Form außerhalb des Systems in sogenannten dynamischen Objekten plaziert. Damit ist das Kernsystem von Formaten und Netzwerktechnologien unabhängig geworden. Verteilte Streams sind dann als spezielles Stream-Format realisiert worden.

Die Analyse wurde mit der funktionalen Modellierung abgeschlossen. Das funktionale Modell zeigt alle wichtigen Datenflüsse und Datentransformationen innerhalb des DMSM-Systems.

Während bei der Analyse weitestgehend geklärt wurde, *was* zu tun ist, unabhängig davon, *wie* es realisiert werden könnte, wurde während des Entwurfs die Frage nach dem *wie* beantwortet. Es wurde die globale Problemlösungsstrategie für die Implementierung des Systems entwickelt. Der Systementwurf umfaßt Entscheidungen über die Organisation des Systems in Teilsystemen sowie wichtige Entscheidungen zur Konzeption und Vorgehensweise, die den Rahmen für die Implementierung bilden. Die wichtigsten Themen waren die Parallelitäten, die Objektkommunikation, die Realisierung der dynamischen Objekte und die Behandlung von Grenzbedingungen.

Mit Hilfe einer Beispielimplementierung wurde es ermöglicht, die Analyse und den Entwurf zu überprüfen. Die Ergebnisse und ihre Bewertung können der Verfeinerung der Modelle in einem neuen Iterationsschritt des Entwurfs dienen.

Teil III

Anhang

Quellcode

Der selbstgeschriebene Quellcode aller zum DMSM-System gehörenden Programme, Module und Datendateien inklusive der Dokumentationen sind auf eine Diskette dieser Diplomarbeit beigelegt. Die Dateien sind als tar-File (mit gzip komprimiert) im MS-DOS-Format auf der Diskette gespeichert.

Der folgende Auszug aus der Verzeichnisstruktur gibt einen Überblick:

Root-Directory

```
DMSMSApplication.cpp                    IIRFilter.cpp
DMSMSApplication.dsp                    IIRFilter.h
DMSMSApplication.dsw                    ScratchPanel.cpp
DMSMSApplication.h                      ScratchPanel.h
DMSMSApplication.rc                     StreamControlDlg.cpp
DMSMSApplicationDlg.cpp                 StreamControlDlg.h
DMSMSApplicationDlg.h
```

Directory of DMSMStreamObjectEditor

```
ChildFrm.cpp                            DMSMStreamObjectEditorDoc.cpp
ChildFrm.h                              DMSMStreamObjectEditorDoc.h
DMSMStreamObjectEditor.cpp              DMSMStreamObjectEditorView.cpp
DMSMStreamObjectEditor.dsp              DMSMStreamObjectEditorView.h
DMSMStreamObjectEditor.h                MainFrm.cpp
DMSMStreamObjectEditor.rc               MainFrm.h
```

Directory of DMSMSCore

Announcement.cpp
Announcement.h
ApplicationType.cpp
ApplicationType.h
Capability.cpp
Capability.h
ClassBroker.cpp
ClassBroker.h
DataDirection.cpp
DataDirection.h
DMSMSCore.cpp
DMSMSCore.def
DMSMSCore.dsp
DMSMSCore.rc
DMSMSCoreAfxExt.h
DMSRCPComm.cpp
DMSRCPComm.h
DMSRCProtocol.h
DMStream.cpp
DMStream.h
DynamicObject.cpp
DynamicObject.h
FileWatchdog.cpp
FileWatchdog.h
GenericStreamControl.cpp
GenericStreamControl.h
GenericStreamTransport.cpp
GenericStreamTransport.h
Host.cpp
Host.h
Inquiry.cpp
Inquiry.h
MediaType.cpp
MediaType.h
ObjectBroker.cpp
ObjectBroker.h
ProcessAddress.cpp
ProcessAddress.h
ProtocolType.cpp
ProtocolType.h
StreamObject.cpp
StreamObject.h
StreamObjectTemplate.cpp
StreamObjectTemplate.h
SubjectObserver.cpp
SubjectObserver.h

Directory of KonnexService

AnnounceCallback.cpp
AnnounceCallback.h
CapabilityAnnounceCallback.cpp
CapabilityAnnounceCallback.h
CapabilityAnnouncement.cpp
CapabilityAnnouncement.h
InquiryCallback.cpp
InquiryCallback.h
InquiryResponse.cpp
InquiryResponse.h
KonnexLayer.cpp
KonnexLayer.h
KonnexService.cpp
KonnexService.def
KonnexService.dsp
KonnexService.h
KonnexService.rc
KonnexServiceAfxExt.h

Directory of RegistrationService

DMStreamManager.cpp
DMStreamManager.h
RegistrationService.cpp
RegistrationService.def
RegistrationService.dsp
RegistrationService.h
RegistrationService.rc
RegistrationServiceAfxExt.h

Directory of TransportService

ConnectionResponse.cpp
ConnectionResponse.h
TransportService.cpp
TransportService.def
TransportService.dsp
TransportService.h
TransportService.rc
TransportServiceAfxExt.h

Directory of TransportService\StreamEnginePcmFile2Tcp

PCMSource.cpp
PCMSource.h
ReadMe.txt
Resource.h
StreamEngineClassAfxExt.h
StreamEnginePcmFile2Tcp.cpp
StreamEnginePcmFile2Tcp.def
StreamEnginePcmFile2Tcp.dsp
StreamEnginePcmFile2Tcp.h
StreamEnginePcmFile2Tcp.rc
TCPSink.cpp
TCPSink.h

Directory of TransportService\StreamEnginePcmIn2Tcp

```
PCMSource.cpp                       StreamenginePcmIn2Tcp.def
PCMSource.h                         StreamenginePcmIn2Tcp.dsp
ReadMe.txt                          StreamenginePcmIn2Tcp.h
Resource.h                          StreamenginePcmIn2Tcp.rc
StreamEngineClassAfxExt.h           TCPSink.cpp
StreamenginePcmIn2Tcp.cpp           TCPSink.h
```

Directory of TransportService\StreamEngineTcp2PcmFile

```
PCMSink.cpp                         StreamEngineTcp2PCMFile.h
PCMSink.h                           StreamEngineTcp2PCMFile.rc
StreamEngineClassAfxExt.h           StreamEngineWav2Tcp.def
StreamEngineTcp2PCMFile.cpp         TCPSource.cpp
StreamEngineTcp2PCMFile.def         TCPSource.h
StreamEngineTcp2PCMFile.dsp
```

Directory of TransportService\StreamEngineTcp2PcmOut

```
IIRFilter.cpp                       StreamEngineTcp2PCMOut.dsp
IIRFilter.h                         StreamEngineTcp2PCMOut.h
PCMSink.cpp                         StreamEngineTcp2PCMOut.rc
PCMSink.h                           StreamEngineWav2Tcp.def
StreamEngineClassAfxExt.h           TCPSource.cpp
StreamEngineTcp2PCMOut.cpp          TCPSource.h
StreamEngineTcp2PCMOut.def
```

Directory of TransportService\StreamEngineWav2Tcp

```
StreamEngineClassAfxExt.h           TCPSink.cpp
StreamEngineWav2Tcp.cpp             TCPSink.h
StreamEngineWav2Tcp.def             WAVSource.cpp
StreamEngineWav2Tcp.dsp             WAVSource.h
StreamEngineWav2Tcp.h
StreamEngineWav2Tcp.rc
```

Directory of TransportService\TransportClassSimpleTCP

```
SimpleTCP.cpp                       TransportClassSimpleTCP.def
SimpleTCP.h                         TransportClassSimpleTCP.dsp
TransportClassAfxExt.h              TransportClassSimpleTCP.rc
TransportClassSimpleTCP.cpp
```

Literaturverzeichnis

[Rum93] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen,
"Objektorientiertes Modellieren und Entwerfen",
Prentice Hall International, ISBN 3-446-17520-2, 1993

[Fro94] Michael Fromme, Lutz Grüneberg,
"Software-Engineering"
Lehrgebiet Rechnernetze und Verteilte Systeme, Universität Hannover, 1994

[Unb90] Rolf Unbehauen,
"Systemtheorie", 5.Auflage,
Oldenbourg, 1990

[Coh95] Reuven Cohen; Yee-Hsian Chang,
"Video-on-Demand session management",
Alto, Calif. : Hewlett Packard Laboratories, 1995
(HP Laboratories technical report ; 95-18)

[Pys95] T. Pyssysalo; L. Ojala.
"High-Level Net Model of a Video on Demand System",
ISBN 951-22-2917-X , 1995

[Che95] Shenze Chen,
"Fibre channel storage interface for video-on-demand servers",
Thapar. - Palo Alto, Calif. : Hewlett Packard Laboratories , 1995
(HP Laboratories technical report ; 95-125)

[Vis94] S. Viswanathan and T., Imielinski,
"Pyramid broadcasting for video on demand service",
New Brunswick, N.J. : Dep. of Computer Science, Rutgers University, 1994

[Fen97] Wu-chi Feng,
"Buffering techniques for delivery of compressed video in video-on-

demand", Kluwer Academic Publishers, ISBN 0-7923-9998-6, 1997

[IEE94] Electronics Division, the Institution of Electrical Engineers,
"Colloquium on High Speed Access Technology and Services Services, Including Video-on-Demand ",
Electronics Division, the Institution of Electrical Engineers. - London, 1994

[Pys95] T. Pyssysalo, L. Ojala,
"High-Level Net Model of a Video on Demand System",
ISBN 951-22-2917-X , Report-Nr.: PB96-184999, 1995

[Cha91] U. Chan,
"Video on demand (VOD) service" : initial consideration,
Research Labs report / M. Chan. - Adelaide : Telecom Australia, 1991

[Ein97] Ralf Einhorn,
"Entwicklung eines Video-Editier-Systems auf Basis verteilter Videoserver",
Diplomarbeit, RVS, Universität Hannover, 1997

[Par93] Parekh, A.,
"A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks", MIT Laboratory for Information and Decision Systems,
Report No. LIDS-TH-2089.

[Cla92] Clark, D., Shenker, S., and L. Zhang,
"Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", Proceedings of ACM SIGCOMM '92

[RFC1521] Borenstein, N., and N. Freed.
"MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies",
RFC 1521, Bellcore, Innosoft, September 1993.
<URL:ftp://ds.internic.net/rfc/rfc1521.txt>

[RFC1590] Postel, J.
"Media Type Registration Procedure",
RFC 1590, USC/Information Sciences Institute, March 1994.
<URL:ftp://ds.internic.net/rfc/rfc1590.txt>

[RFC1363] Partridge, J.
"A Proposed Flow Specification",
RFC 1363, Sept. 1992.
<URL:ftp://ds.internic.net/rfc/rfc1590.txt>

[RFC1889] H. Schulzrinne, R.Frederick, V. Jacobson
"A Transport Protocol for Real-Time Applications",
RFC 1889, Jan. 1996.
<URL:ftp://ds.internic.net/rfc/rfc1889.txt>

[SPEC95] Zhang, L., Braden, R., Estrin, D., Herzog, S., and S. Jamin,
"Resource ReSerVation Protocol (RSVP) – Version 1 Functional"
Specifikation, ftp://ftp.ietf.org/internet-drafts/draft-ietf-rsvp-spec-16.txt

[RSVP93] Zhang, L., Deering, S., Estrin, D., Shenker, S., and D. Zappala,
"RSVP: A New Resource ReSerVation Protocol",
ftp://parcftp.xerox.com/pub/net-research/rsvp.ps.Z , September 1993.

[RFC1633] Braden, R., Clark, D., and S. Shenker,
"Integrated Services in the Internet Architecture: an Overview"
RFC1633, ISI, MIT, and PARC, June 1994
http://ds.internic.net/rfc/rfc1633.txt

[RFC768] Postel, Jonathan B.
"User Datagram Protocol", 1980
http://ds.internic.net/rfc/rfc768.txt

[RFC1112] Deering, Steve
"Host Extensions for IP Multicasting", 1989
http://ds.internic.net/rfc/rfc1112.txt

[TANE96] Andrew S. Tanenbaum,
"Computer networks"
3. Ed. - Upper Saddle River, NJ, Prentice Hall, 1996. - XVII
(Prentice Hall international editions), Previous ed.: 1988
ISBN 0-13-349945-6

[Gerth94] W. Gerth,
"Echtzeitdatenverarbeitung",
Vorlesungsskript: Institut für Regelungstechnik Universität Hannover, 1994

[Tie89] UlrichTietze, Christoph Schenk,
"Halbleiter Schaltungstechnik", 9. Auflage,
Springer-Verlag, ISBN 3-540-19475-4, 1989

[Sme97] Ed Smetak, Jean Caputo,
"Dynamisch änderbare Objekte"
Microsoft System Journal, B 30751, Heft 6, Nov./Dez. 1997

[Ste90] W. Richard Stevens,
"UNIX Network Programming"
Prentice Hall, ISBN 0-13-949876-1, 1990

[Qui96] Bob Quinn, Dave Shute,
"Windows sockets network",
Addison-Wesley, ISBN 0-201-63372-8, 1996.

[Bon96] Pat Bonner,
"Network programming with Windows Sockets",
Prentice Hall PTR, ISBN 0-13-230152-0, 1995-1996

[Bro95] Kraig Brockschmidt,
"Inside OLE", 2nd ed.,
Microsoft Press, ISBN 1-55615-843-2, 1995