Multimedia Design Pattern

Erweiterungen und Umsetzungen des „Pipes and Filters“-Patterns für Multimedia-Komponentensysteme

Philipp Merkel
Seminar Forschungstrends im Bereich Verteilte Systeme
Universität Ulm
Wintersemester 2008/2009
philipp.merkel@uni-ulm.de

ABSTRACT

Um im Multimediabereich die Wiederverwendbarkeit von Programmcode zu erhöhen, gibt es viele Projekte mit dem Ziel, Komponentenframeworks zu entwickeln, mit denen komplexe Multimediasysteme aus einfachen Komponenten aufgebaut werden können. Diese Komponenten sollen austauschbar und so in möglichst vielen anderen Projekten wiederverwendbar sein. Neuere Ansätze sehen zudem meist vor, dass die verschiedenen Komponenten auch verteilt auf verschiedenen Knoten in einem Netzwerk betrieben werden können. Als Grundlage verwenden diese Systeme das Design Pattern *Pipes And Filters*, erweitern es auf vielfältige Weise und entwickeln konkrete Umsetzungskonzepte. Hiermit soll die dynamische, teilweise automatisierte Zusammenstellung von Komponenten, die verteilte Installation von Komponenten im Netzwerk sowie die Datenübertragung und Signalisierung im lokalen und entfernten Fall ermöglicht werden. Dieses Paper stellt das Design Pattern *Pipes And Filters* sowie die angesprochenen Erweiterungen vor und gibt Anwendungsbeispiele.

Stichwörter

Komponenten, Multimedia, Frameworks, Pipes and Filters, Netzwerk, Verteilte Systeme

1. EINLEITUNG

Die Wiederverwendbarkeit von Programmkomponenten ist ein immer wichtiger werdendes Ziel bei der Entwicklung von Software. Um das Management von stets komplexer werdenden Softwaresystemen überhaupt erst zu ermöglichen, sind mehr und mehr Anwendungen – vom Datei-Manager bis zur Entwicklungsumgebung – aus austausch- und wiederverwendbaren Softwarekomponenten aufgebaut.

Auch im Multimediabereich gibt es Frameworks, insbesondere zur Audio- und Videodecodierung, die den Aufbau von Systemen erleichtern sollen. Hier sind etwa Microsoft DirectShow [1], Apple QuickTime [2], GStreamer [3] oder Xine [4] zu nennen. Allerdings sind dies keine leichtgewichtigen, beliebig kombinierbaren Komponenten, sondern große, teilweise geschlossene und nicht erweiterbare Bibliotheken. Auch für die Übertragung von Multimediadaten über Netzwerke gibt es solche Frameworks. Jedoch ist es hier häufig der Fall, dass für verschiedene Dienste wie *Voice over Internet Protocol* (VoIP) oder *Video on Demand* (VoD) verschiedene, komplett unterschiedliche Bibliotheken herangezogen werden müssen, die sich nur mit großem Programmieraufwand miteinander und mit den Decodierungsbibliotheken kombinieren lassen.

Was die Weiterverarbeitung der Multimediadaten angeht, sieht es ähnlich aus. Auch hier müssen oft unterschiedliche Verarbeitungsbibliotheken mit verschiedensten Interfaces kombiniert und durch eigene Algorithmen ergänzt werden.

Daher gibt es zahlreiche Forschungsprojekte mit dem Ziel, komponentenbasierte Frameworks zu schaffen, um den Aufbau komplexer multimedialer Anwendungen aus wiederverwendbaren Komponenten – basierend auf standardisierten Schnittstellen und mit Unterstützung durch spezielle Middlewaresysteme – zu ermöglichen. Dahinter steckt auch die Idee eines *Komponentenmarkts*, auf dem solche, von verschiedensten Herstellern entwickelte Softwarekomponenten angeboten und vom Benutzer (gegen Geld oder kostenlos) „erworben“ werden können, um sie dann in eigene Systeme zu integrieren.

Neuere Konzepte gehen noch weiter: Komponenten werden nicht nur zur späteren lokalen Verwendung beim Kunden angeboten, sondern als Dienstleistungen, die sich über das Internet nutzen lassen: Dienstanbieter installieren Service-Komponenten auf ihren Servern, die dann von Privat- oder Geschäftskunden bei Bedarf aufgerufen werden können, um ihre Multimediadatenströme zu verarbeiten.

Im Folgenden sollen nun Konzepte für solche Multimedia-Komponentenframeworks vorgestellt werden. Zunächst werden einige Beispiele für mögliche praktische Anwendungen solcher Systeme aufgezeigt, anschließend auf das Design Pattern eingegangen, das allen Konzepten zugrunde liegt. Danach werden Erweiterungen und Umsetzungskonzepte für dieses Pattern vorgestellt, die die spezifischen Probleme des Anwendungsbereichs adressieren, bevor eine abschließende Zusammenfassung mit Ausblick folgt.

2. ANWENDUNGSBEISPIELE

Komponentenframeworks können für die Realisation zahlreicher Multimediasysteme herangezogen werden, um neue Dienste zu ermöglichen und bestehende zu verbessern.

Als einfaches System, das auf einem solchen Framework aufbaut, ist ein VoD-Service vorstellbar. Auf der Serverseite können unterschiedliche Komponenten für die verschiedenen

Auflösungen, Qualitätsstufen und Codecs, in denen Filme ausgeliefert werden können, sowie für verschiedene Übertragungsprotokolle genutzt werden. Ist auch die Clientanwendung komponentenbasiert aufgebaut, können auch hier Komponenten für die Decodierung von in verschiedenen Formaten codierten Videos eingesetzt werden.

Beide Seiten können durch das Hinzufügen von neuen Komponenten erweitert werden, um neue Codecs und Protokolle zu unterstützen. Bei verteilten Komponentenframeworks können zudem auf weiteren Knoten zwischen Client und Server neue Komponenten installiert werden, um die Multimediadaten z. B. in für Mobiltelefone taugliche Formate umzucodieren.

Als komplexeres Anwendungsbeispiel ist ein VoIP-Telefonsystem denkbar, das aus einfachen Komponenten aufgebaut ist, die u. a. für den Rufaufbau, das Einspielen von Rufzeichen, die Datenübertragung, die Codierung von Audiodaten ins G.729-Format und die Decodierung der Daten aus diesem Format zuständig sind.

Ein solches System kann dann bei Bedarf (entweder durch den Dienstanbieter oder den Nutzer) erweitert werden. So kann eine Komponente hinzugefügt werden, die außerdem die Decodierung von MP3-Daten erlaubt, oder eine, die bei Abwesenheit das Aufzeichnen einer Nachricht ermöglicht. Die Rufzeichenkomponente kann durch eine andere ausgetauscht werden, die statt Pieptönen Musik einspielt, während man wartet, dass das Gegenüber abhebt. In einigen Jahren, wenn die Spracherkennungs-, Sprachsynthetisierungs- und Übersetzungstechnologie dies erlaubt, kann dann eine zusätzliche Komponente integriert werden, die automatisch und in Echtzeit Gespräche zwischen verschiedensprachigen Teilnehmern übersetzt – ohne dass es hierfür notwendig ist, die anderen Komponenten zu verändern.

Bei einem verteilten Komponentensystem, das die Verwendung von Diensten auf entfernten Servern erlaubt, können die Komponenten für aufwändige Dienste wie den Übersetzungsdienst statt auf dem Rechner oder Mobiltelefon des Endbenutzers auf einem Server der Telefongesellschaft laufen – oder bei einem externen Dienstleister, der den Übersetzungsdienst unabhängig vom VoIP-Anbieter bereitstellt.

3. PIPES AND FILTERS

Um verschiedenste Codier-/Decodier-, Verarbeitungs- und Übertragungssysteme unter einem Grundkonzept mit einheitlichen Schnittstellen zu vereinen, bauen die im Folgenden vorgestellten Konzepte für Multimedia-Komponentenframeworks auf einem Design Pattern auf, das sich *Pipes and Filters* nennt.

Pipes and Filters ist ein allgemeines Pattern für die Verteilung der Verarbeitung von Datenströmen auf mehrere sogenannte Filter. Die Filter sind in einer Kette aufeinanderfolgender Verarbeitungsschritte, der sog. *Pipeline*, angeordnet. Die Verbindungen zwischen den einzelnen Filtern werden als *Pipes* bezeichnet.

Ein *Filter* (Abbildung 1) ist eine konzeptionelle Software-Komponente, die einen Eingang besitzt, durch den sie einen Datenstrom empfängt, sowie einen Ausgang, durch den sie

(a) Einfacher Filter mit Ein- und Ausgang

(b) Datenquelle

(c) Datensenke

Abbildung 1: Die verschiedenen Typen von Filtern

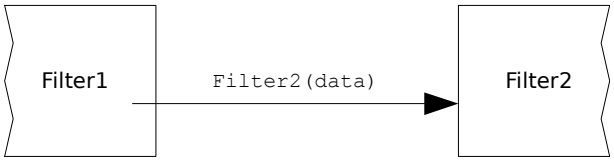
den veränderten Datenstrom an eine andere Komponente weiterleiten kann. Der Filter übernimmt hierbei eine eindeutig abgegrenzte Aufgabe, ohne zu berücksichtigen, wo die Daten, die er verarbeitet, herkommen und was mit ihnen geschieht, nachdem er sie weitergeleitet hat. Eine typische Aufgabe für einen Filter ist z. B. die Umcodierung von Daten aus einem Format in ein anderes.

Spezielle Filter sind sog. *Datenquellen* [5], die stets am Anfang einer Pipeline stehen und neue Daten in das System einbringen. Dabei können sie diese Daten entweder selbst erzeugen oder von einer externen Quelle, die nicht Bestandteil der Pipeline ist, beziehen. Eine mögliche Quelle kann etwa ein an den Rechner angeschlossenes Eingabegerät sein, wie eine Tastatur oder Videokamera, oder ein Stream aus einem Netzwerk.

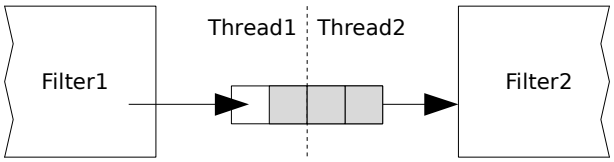
Analog wird als letzter Filter einer Pipeline stets eine *Datensenke* [5] verwendet, die keinen Ausgang, sondern nur einen Eingang besitzt. Sie leitet die Daten üblicherweise an ein anderes, von der Pipeline unabhängiges Ziel weiter, indem sie sie etwa auf dem Bildschirm ausgibt oder über das Netzwerk verschickt.

Einer *Pipe* (Abbildung 2) liegt das Konzept einer Leitung zugrunde, durch die Daten von Filter zu Filter geleitet werden. Eine Pipe verbindet somit stets den Ausgang eines Filters mit dem Eingang eines anderen.

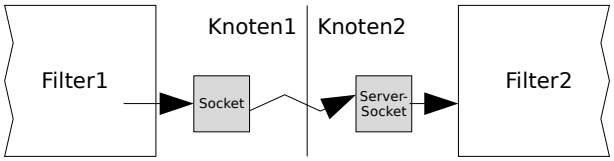
Wie diese Verbindung zwischen den Filtern realisiert wird, schreibt das Design Pattern nicht vor. Eine Möglichkeit – der *Push*-Ansatz [5] – ist, dass ein Filter den ihm in der Kette nachfolgenden in Form einer Methode aufruft, sobald er neue Daten hat, und ihm diese Daten als Argument übergibt. Der umgekehrte *Pull*-Ansatz [5] sieht vor, dass ein Filter, wenn



(a) Methodenaufruf



(b) Lokaler Puffer



(c) Transport über das Netzwerk

Abbildung 2: Verschiedene Typen von Pipes

er neue Daten verarbeiten kann, den ihm in der Kette vorgelagerten Filter als Funktion aufruft, die dann die Eingabedaten zurückliefert. In diesen beiden Ansätzen ist die Pipe als tatsächliches Konstrukt nicht vorhanden, die Daten werden stets direkt übergeben. Allerdings ist es mit diesen beiden Umsetzungen schwierig, Nebenläufigkeit zu realisieren.

Alternativ kann jeder Filter als in einem eigenen Thread laufendes Programm aufgebaut sein, das Daten aus einem Puffer liest, in den sie vom vorhergehenden Filter geschoben werden, und sie anschließend in einen weiteren Puffer schiebt, um sie an den nächsten Filter weiterzuleiten. In diesem Fall sind alle Filterthreads nebenläufig aktiv und können Daten verarbeiten, sobald in ihrem Eingangspuffer genügend vorliegen. Die Pipes sind hier durch die Puffer als tatsächliche Objekte modelliert.

Eine Pipe kann durchaus auch über Rechnergrenzen hinweg aufgebaut sein. In diesem Fall müssen die Daten über Netzwerkprotokolle übertragen werden. In den ersten beiden genannten Ansätzen bietet sich hierfür ein *Remote Procedure Call* (RPC) [6] an, im dritten können Sockets [7] verwendet werden, die durch ein beliebiges strombasiertes Protokoll verbunden sind.

Herausragendes Merkmal des Pipes-And-Filters-Patterns ist, dass ein Filter die eintreffenden Daten inkrementell verarbeiten und weiterleiten kann, ohne zuvor die gesamte zu verarbeitende Datenmenge empfangen und puffern zu müssen (vgl. [5]). Nachgelagerte Filter können somit bereits mit der Weiterverarbeitung beginnen, sobald sie die ersten Daten empfangen, und müssen nicht warten, bis die Vorgängerfilter die gesamte Eingabe verarbeitet haben.

Außerdem lassen sich mit Hilfe dieses Pattern komplexe

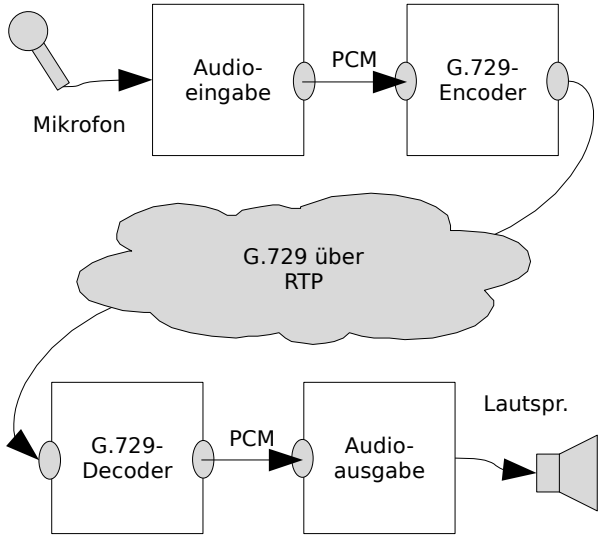


Abbildung 3: Einfache Pipeline im VoIP-Beispiel

Verarbeitungsschritte durch die Kombination kleiner, vergleichsweise einfacher Filter realisieren, die bei Bedarf durch andere Filter ausgetauscht werden können, um das Programm anzupassen.

Die inkrementelle Verarbeitung der Daten und die Kombinierbarkeit der Filter sind der Grund, dass sich das Pipes-And-Filters-Konzept als passende Grundlage für Mulimedia-Komponentensysteme erwiesen hat: Schließlich liegen Multimediadaten meist in großen Mengen vor, die nur inkrementell sinnvoll verarbeitet und übertragen werden können, und die Kombinations- und Austauschmöglichkeit von Filtern erhöht die Wiederverwendbarkeit von Code und die Adaptionsfähigkeit der Software. Ein Beispiel für die Verwendung von Pipes and Filters in unserem VoIP-Beispiel ist in Abbildung 3 dargestellt: Die Daten werden aus einem Mikrofon in einen Encoder geleitet, der sie komprimiert. Über RTP werden sie zum Empfänger gesendet, wo sie wieder decodiert und über einen Lautsprecher ausgegeben werden.

3.1 Tee-and-Join-Pipeline-Systeme

Eine interessante Erweiterung des generischen Pipes-And-Filters-Patterns stellen *Tee-and-Join-Pipeline-Systeme* [5] dar. Hierbei können die Filter jeweils mehrere Ein- und Ausgänge besitzen und somit statt zu einer Kette zu einem Netzwerk kombiniert werden.

Dadurch ist es möglich, die Verarbeitung von komplexen Daten aufzuspalten und später wieder zusammenzufügen oder auch Daten aus verschiedenen Quellen zu kombinieren bzw. an verschiedene Senken zu verteilen.

In der Multimediaverarbeitung lässt sich dieses Konzept nutzen, um z. B. in einem Demultiplex-Filter Videodaten in Bild- und Tonspur zu trennen und von verschiedenen Filtern weiterverarbeiten zu lassen. Für die Übertragung über das Netzwerk können die beiden Spuren durch einen Multiplex-Filter wieder zusammengefügt werden (vgl. Abbildung 4).

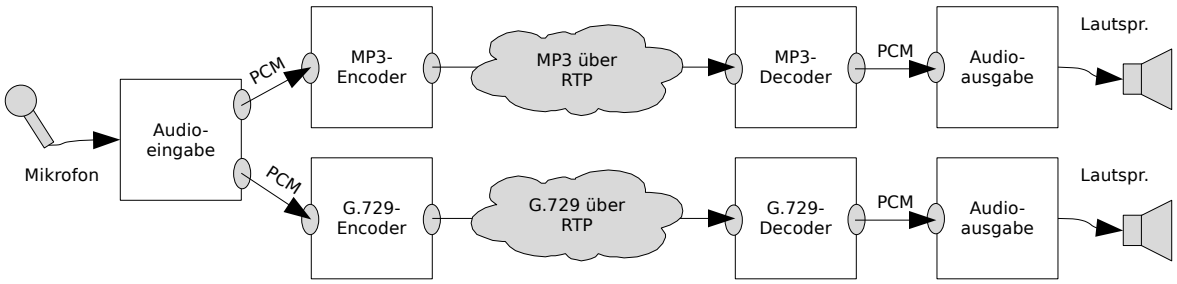


Abbildung 5: Tee-and-Join-Pipeline-System für Telefonkonferenz

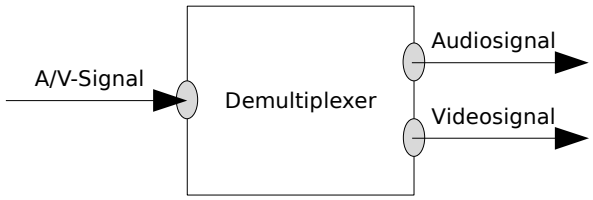
Auch ist es so möglich, Daten über ein Netzwerk nach der Verarbeitung an mehrere Empfänger zu schicken, für die sie auf unterschiedliche Weise weiter aufbereitet werden sollen. Letzteres ist in Abbildung 5 zu sehen, wo ein Ausschnitt aus der Infrastruktur für eine Telefonkonferenz als Tee-and-Join-Pipeline-System zu sehen ist. Die Daten werden hier nach der Eingabe über zwei Kanäle zu den beiden anderen Konferenzteilnehmern geschickt.

4. ERWEITERUNGEN

Für dynamische Komponentensysteme muss dieses Design Pattern in einheitliche Schnittstellen und eine vorbereitete Kommunikations-Infrastruktur für die Komponenten umgesetzt werden. Doch dies alleine reicht nicht aus: Das System muss auf das dynamische Hinzufügen und Entfernen von Komponenten reagieren und sich an veränderte Übertragungsbedingungen und Nutzeranforderungen anpassen können, ggf. für die Verteilung der Komponenten auf verschiedene Knoten sorgen sowie Signalisierung zwischen den Komponenten unterstützen.

Daher wurde das Design Pattern *Pipes and Filters* in einer Reihe von Projekten in verschiedene Richtungen ergänzt und mit Implementierungsaspekten angereichert. Die unterschiedlichen Erweiterungen sollen im Folgenden vorgestellt werden.

(a) Multiplexer



(b) Demultiplexer

Abbildung 4: Multiplexer und Demultiplexer für Audio-/Videodaten

4.1 Verteilte Installation

Viele neuere Multimedia-Komponentensysteme wurden explizit entwickelt, um die verteilte Verarbeitung von Multimediadaten im Netzwerk mit weitestgehend unabhängigen, dienstartigen Komponenten auf verschiedenen Rechnern zu ermöglichen. Hierbei werden die einzelnen Komponenten von den jeweiligen Anwendern bzw. Dienstbetreibern auf Servern installiert, auf denen sie dann ihre Dienste anderen Komponenten im Netzwerk zur Verfügung stellen.

So sehen z. B. Wijnants et al. in [8] vor, dass Filter als Plugins auf einem Proxy-Server installiert werden, wo sie sich auf Netzwerkprotokollebene in Datenströme einklinken und sie verändern können. In unserem VoIP-Beispiel kann der Telefonieanbieter für den Übersetzungsdienst so z. B. ein Plugin installieren, das die über den Proxy-Server übertragenen G.729-Daten decodiert und in decodierter Form weiterschickt, ein weiteres Plugin kann auf den Rohdaten eine Spracherkennung durchführen, ein anderes die Übersetzung durchführen, ein weiteres die Sprachsynthetisierung und schließlich eines wieder die Rückcodierung in G.729. Zumindest das Decodierungs- und das Spracherkennungsplugin sowie das Sprachsynthetisierungs- und Codierungsplugin sollten jeweils auf dem selben Proxy-Server laufen, um die Übertragung der Rohdaten über das Netz zu vermeiden.

In MSODA [9] läuft jeder Service, also jede Komponente, in einer virtuellen Maschine (VM) mit eigenem Betriebssystem, um andere Services nicht zu beeinflussen. Hier sollten im Übersetzungsbeispiel die Komponenten, zwischen denen Rohdaten übertragen werden, nicht nur auf einem Server, sondern in einem Service zusammengefasst werden, um die kostspielige Übertragung über VM-Grenzen hinweg zu vermeiden.

Bei Wagner und Keller [10] sowie Scholz et al. [11] ist vorgesehen, dass die Komponenten ihre Dienste in Form von Web-Services anbieten. Hier müssen nun die Komponenten zur VoIP-Übersetzung auf einem oder mehreren Web-Servern installiert werden, die unabhängig von den Servern des VoIP-Dienstanbieters betrieben werden können. Auch hier sollten die Dienste so installiert werden, dass keine Übertragung von Rohdaten zwischen verschiedenen Web-Services nötig ist.

4.2 Kombination

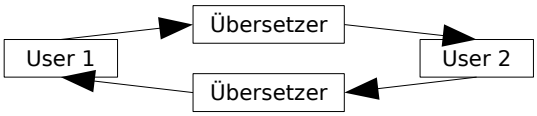
Sind die einzelnen Komponenten installiert, können sie dynamisch kombiniert werden, um daraus neue Dienste zusammenzustellen.

Die *Network Integrated Multimedia Middleware* (NMM) [12] sieht einen Kontrollgraphen vor, der die Filter und ihre Verbindungen enthält. Hierbei soll der Benutzer auf einer höheren Abstraktionsebene die gewünschte Funktionalität beschreiben können, woraus das System dann passende Kontrollgraphen generiert und im laufenden System anhand von Kriterien wie der Qualität des beim Nutzer ankommenden Signals automatisch den besten aussucht. Im VoIP-Beispiel könnte so automatisch der Codec ausgewählt werden, der von den bei allen Gesprächspartnern verfügbaren Codecs die beste Übertragungsqualität bei möglichst geringem Bandbreitenbedarf liefert.

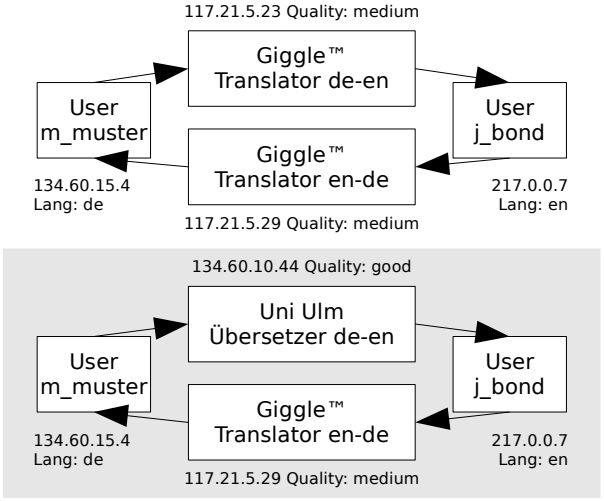
Xu und Jiang [9] gehen ähnliche Wege. Auch hier beschreibt der Benutzer mittels eines speziellen Programmierinterfaces die für einen Service benötigten Komponenten sowie Daten- und Zeitabhängigkeiten. Das System überprüft automatisch die Realisierbarkeit der Komposition, also z. B. ob alle verbundenen Komponenten kompatible Datenformate unterstützen. Ist diese Überprüfung erfolgreich, generiert auch dieser Ansatz konkrete Konfigurationen, wobei auf zusätzliche Randbedingungen wie die verfügbare Bandbreite im Netzwerk oder spezifische Eigenschaften der Benutzer geachtet wird. Anschließend werden die benötigten Typen von Komponenten (hier *Basic Services* genannt) auf die Knoten im Netzwerk und die in den darauf laufenden virtuelle Maschinen verfügbaren Basic Services abgebildet und die beste Abbildung schließlich gewählt. Abbildung 6 zeigt dies am Beispiel des VoIP-Systems: Der Benutzer gibt als allgemeine Konfiguration an, dass zwischen den Gesprächspartnern in beide Richtungen eine Übersetzung stattfinden soll. Das System erzeugt daraufhin automatisch Konfigurationen, die passende Übersetzungskomponenten für die Sprachen der Teilnehmer enthalten. Daraus wird dann die beste Konfiguration gewählt, im Beispiel die zweite, da die Übersetzungsqualität hier besser ist.

Wagner und Kellerer [10] gehen bei der Abstraktion noch weiter: hier wird die Semantik von Komponenten (die in diesem Fall in Form von Web-Services realisiert sind) in einer Beschreibungssprache namens OWL-S definiert. Eine OWL-S-Beschreibung besteht aus drei Teilen: Zum einen wird die Semantik des Services beschrieben, zum anderen die Funktionsweise in Form eines Workflows angegeben und als drittes die Schnittstelle definiert. Semantikunterstützte Software kann aus diesen Web-Service-Beschreibungen nun automatisch die Komponenten auswählen, die bei Kombination die vom Benutzer gewünschte Funktionalität bieten.

Beim *Distributed Feature Composition*-Konzept (DFC) [13] (auf dem auch der Ansatz von Cheung und Purdy [14] aufbaut) hingegen muss der Administrator weitestgehend manuell konfigurieren, in welcher Reihenfolge die verschiedenen Filter, die in diesem Fall Features eines Telefonsystems darstellen, ausgewählt werden dürfen. Allerdings muss nur eine partielle Ordnung angegeben werden (vgl. Abbildung 7(a)). Außerdem enthält das System eine Liste der verschiedenen Features, bei denen sich jeder Teilnehmer des Telefonsystems angemeldet hat (Abbildung 7(b)). Es generiert



(a) Vom Benutzer angegebenes Modell



(b) Vom System erzeugte Konfigurationen

Abbildung 6: Telefongespräch mit Übersetzung

dann für jede Signalisierungsnachricht eine totale Ordnung auf diesen Features, die die partielle Ordnung erfüllt. In unserem VoIP-Beispiel ist es etwa möglich, anzugeben, dass das Feature „Übersetzer“ auf der Empfängerseite vor dem Feature „Mailbox“ angeordnet sein muss, damit die Nachrichten übersetzt werden, bevor sie auf die Mailbox gelangen. Hat ein Kunde beide Features aktiviert, sorgt das DFC-Verfahren dafür, dass sie immer in dieser Reihenfolge abgearbeitet werden.

Bei Gibbs [15], Posnak et al. [16] und in anderen Ansätzen muss der Entwickler die Komponenten selbst im Programmcode zusammensetzen, indem er den Eingang eines Filters manuell beim Ausgang eines anderen Filters registriert und sie so verbindet. Bei Posnak et al. [16] beispielsweise werden die Ausgabeschnittstellen durch `Port`-Objekte dargestellt, die mittels eines Aufrufs der Methode `attach(`*`Filter`*`)` mit anderen Filtern verbunden werden können. Eine automatische Adaption an Benutzeranforderungen oder Netzwerkbandbreiten ist hierbei nicht vom Framework vorgesehen, sondern muss vom Entwickler bei Bedarf selbst umgesetzt werden.

Scholz et al. gehen bei ihrem WS-AMUSE [11]-Konzept einen weiteren Weg: zwar muss hier auch der Entwickler eines auf dem Komponentenmodell basierenden Systems die Komponenten (hier wieder Web-Services) selbst verbinden, allerdings nicht im Programmcode, sondern in Form eines Zustandsautomaten. Dieser wird anschließend automatisch in einen Prozess der *Business Process Execution Language* (BPEL) umgewandelt. Ein solcher Prozess besteht aus einer Kombination von Web-Service-Aufrufen und kann durch eine BPEL-Engine ausgeführt werden.

4.3 Kooperation

Wenn alle Komponenten lokal vorliegen und im Programm direkt miteinander verbunden werden, erfolgen Signalisierungsaufrufe zwischen Komponenten meist direkt durch Methodenaufrufe.

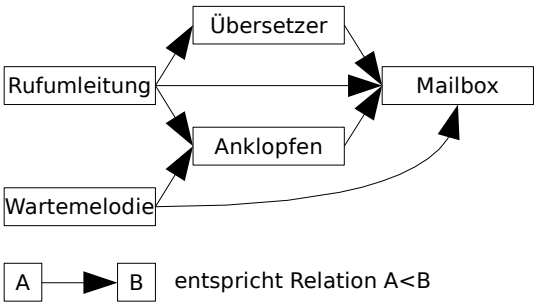
So etwa bei dem Konzept der *Adaptive Pipeline* [16]: Hier kann an einem Filterobjekt eine Methode namens `Control(`*`ControlInformation`*`)` aufgerufen werden, um es zu steuern. Außerdem kann ein Filter eine `invokeControl (`*`ControlInformation`*`)`-Methode an seinem `Port`-Objekt aufrufen, um beim ihm in der Filterkette nachfolgenden Objekt einen `Control`-Methodenaufruf auszulösen. Die Methodenaufrufe folgen damit also konzeptionell den Pipes aus dem Pipes-And-Filters-Pattern.

Auch bei Gibbs [15] können Filter über Methodenaufrufe gesteuert werden, außerdem können Objekte Events auslösen, die von anderen Objekten aufgefangen werden können.

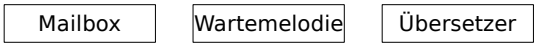
Im verteilten Fall gibt es Ansätze, die diese Semantik möglichst genau übernehmen möchten und daher entfernte Methodenaufrufe zur Kooperation verwenden. Im NMM-System [12], das auf Basis von CORBA [17] aufgebaut ist, geschieht dies auf klassische Weise über Proxyobjekte. Wird an einem solchen Proxyobjekt eine Kontrollmethode aufgerufen, wird ein *Event* erzeugt, das über das Netzwerk an das eigentliche Filterobjekt weitergeleitet wird, wo es via Reflection wieder in einen Methodenaufruf umgewandelt wird. Allerdings können solche Events alternativ auch innerhalb der durch die Pipes fließenden Datenströme übertragen werden. Zudem ist es, wenn zwei Komponenten auf dem selben Rechner laufen, auch möglich, direkt Methoden aufzurufen, statt den Umweg über die Proxyobjekte zu gehen.

Beim Web-Service-basierten WS-AMUSE [11] erfolgt die Signalisierung über das nachrichtenbasierte Protokoll *SOAP*. Durch SOAP-Nachrichten wird der Zustandsautomat, durch den ein Multimediadienst hier dargestellt wird, von Zustand zu Zustand weitergeschaltet. Da Web-Services ein typisches Client-Server-System darstellen und somit kein Rückkanal vom Service zum Client besteht, können keine Events direkt an den Client gesendet werden. Um dennoch Nachrichten vom Service empfangen zu können, müssen Clients spezielle Anfragen schicken, die dann auf dem Server so lange gehalten werden, bis ein Ereignis eintritt. In diesem Fall wird die Nachricht an den Client als Antwort auf die Anfrage zurückgeschickt. Anschließend – sowie im Fall eines vorzeitigen Verbindungsabbruchs – muss der Client sofort eine neue Anfrage schicken, die dann auf dem Server bis zum nächsten Ereignis gehalten wird.

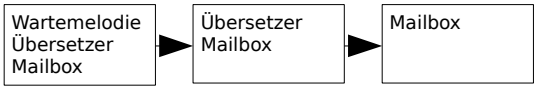
Bei MultiTEL [18], einem weiteren Multimedia-Komponentenframework, kooperieren die Filter nicht durch direkte Aufrufe miteinander. Sie sind reine datenverarbeitende Elemente, die nichts von den anderen im System vorhandenen Komponenten wissen. Stattdessen wird eine Komponente aufgerufen, indem sie eine Nachricht erhält, die sie verarbeitet, um anschließend ein Ereignis auszulösen, das von sog. *Konnektoren* aufgefangen werden kann. Diese leiten es dann an andere Komponenten weiter. Ein Konnektor stellt die Abstraktion eines Kommunikationsprotokolls zwischen Komponenten dar, das unabhängig von den Komponenten ist, die es verbindet. Ein Konnektor kann ohne Änderung des



(a) Partielle Ordnung auf den verfügbaren Features



(b) Von einem Teilnehmer ausgewählte Features



(c) Daraus erstellte Routingliste

Abbildung 7: Kombination von Filtern und Routing von Nachrichten bei Distributed Feature Composition

Komponentencodes ausgetauscht werden, wenn ein anderes Protokoll verwendet werden soll. Die Austauschbarkeit setzt voraus, dass die Konnektoren und die durch sie verbundenen Komponenten in der Lage sind, die selben Ereignisse zu empfangen bzw. zu versenden.

In anderen Systemen werden – insbesondere wenn die einzelnen Komponenten nicht fest „verdrahtet“ sind – Signalisierungsnachrichten zwischen Komponenten indirekt übergeben.

Wie bereits im Abschnitt *Kombination* erklärt, sind in DFC [13] und dem darauf aufbauenden VoIP-System aus [14] nur die Liste der Komponenten, für die sich die Gesprächspartner angemeldet haben, sowie eine partielle Ordnung auf allen Komponenten gegeben. Für den Gesprächsaufbau und andere Signalisierungsaufgaben werden nun auf Grundlage dieser Angaben durch einen im System integrierten *Router* Nachrichten durch die verschiedenen Filter geleitet. Dabei wird zu Beginn aus den angemeldeten Features unter Berücksichtigung der partiellen Ordnung eine totale Ordnung, die sog. *Routingliste*, aufgebaut, an deren erstes Element die Nachricht geleitet wird, wie in Abbildung 7(c) dargestellt ist. Verändert diese Komponente nun das Ziel der Nachricht, wird die Routingliste neu berechnet, ansonsten wird die Nachricht an den nächsten Filter in der Liste weitergeleitet. Dies wird fortgesetzt, bis die Routingliste leer ist und die Nachricht somit ihr endgültiges Ziel erreicht hat. Dadurch ist es den einzelnen Filtern also möglich, durch Verändern der Nachrichten-Zieladresse das Routing zu beeinflussen und so z. B. in einem VoIP-System einen Anruf an einen anderen Anschluss weiterzuleiten, wenn der eigentliche Empfänger nicht verfügbar ist.

4.4 Datenfluss

Wie im vorhergehenden Abschnitt gezeigt wurde, erfolgt die Signalisierung entweder direkt zwischen den Komponenten, oder indirekt durch Events oder Routing. Der Fluss der eigentlichen Multimediadaten durch die Komponenten erfolgt über die Pipes aus dem Pipes-And-Filters-Pattern. Die Umsetzungen dieser Pipes in den verschiedenen Systementwürfen sind vielfältig.

Die klassische Architektur von Kommunikations- und Datenübertragungssystemen sieht vor, dass Signalisierung und Datenfluss komplett getrennt sind. Über die Signalisierung wird eine Verbindung zwischen den beteiligten Systemen (z. B. Gesprächspartner bei einem Telefonsystem oder Client und Server bei einem VoD-Dienst) aufgebaut bzw. manipuliert, über die dann die Daten übertragen werden.

Dieses Konzept wird in seiner reinsten Form beim DFC-System [13] angewendet: Während eine initiale Rufaufbaunachricht mittels des DFC-Routingalgorithmus durch die verschiedenen Komponenten geleitet wird, wird zwischen den Komponenten, die die Nachricht passiert, eine feste Sprachverbindung aufgebaut, durch die anschließend Daten fließen können. Diese passieren somit die Filter in der Reihenfolge, in der die Aufbaunachricht sie besucht hat. Werden weitere Nachrichten über einen anderen Weg geroutet, wird die Verbindung daraufhin adaptiert. Durch eine weitere Nachricht kann die Verbindung wieder abgebaut werden. Konzeptionell erfolgt hier die Kommunikation also durch feste Pipes, die durch das Routen der Signalisierungsnachrichten „verlegt“ werden.

Auch das WS-AMUSE-System von Scholz et al. [11] sieht vor, dass über die Web-Services nur die Signalisierung erfolgt, während die Kommunikationsdaten über klassische Streamingprotokolle wie RTP übertragen werden.

Das System von Wijnants et al. [8] ist sogar ein reines Datenfluss-Protokoll: Es sieht keinerlei Signalisierung vor. Wird sie dennoch benötigt, muss sie extern erfolgen. Hier fließen die Daten über klassische Transportprotokolle über das Netzwerk, und werden „unterwegs“ durch Proxy-Server direkt auf Netzwerkprotokollebene manipuliert.

Andere Systeme vermischen Datenübertragung und Kooperation, indem sie die Daten und Signalisierungsnachrichten auf die selbe Weise, also beide durch die Pipes, übertragen.

Bei der *Adaptive Pipeline* [16] etwa werden auch die Daten zwischen den Filtern durch Methodenaufrufe übertragen. Statt der Methode `Control`, die für die Signalisierung verwendet wurde, wird hierfür die Methode `Input` an einem Filter aufgerufen, um ihm Daten zu übergeben. Der Filter kann dann die veränderten Daten durch Aufruf der Methode `Output` an seinem `Port`-Objekt an den nächsten Filter weiterschicken. Die Daten, die dabei als Argumente der Methodenaufrufe übergeben werden, sind beliebig und hängen nur vom Typ des Ports ab.

Auch bei der NMM [12] sind Signalisierung und Datenfluss in einem Konzept vereint. Die Filter besitzen als Ein- und Ausgabeschnittstellen *Jacks*, durch die zwei Arten von Nachrichten transportiert werden können: *Buffer Messages* werden zur Übertragung von Multimediadaten verwendet. Für Signalisierungsnachrichten können *Composite Events* verwendet werden, die ebenfalls über die Jacks transportiert werden. Buffer Messages können bei einem Filter gepuffert werden, bis genügend Daten für die Verarbeitung vorliegen, während Composite Events direkt bearbeitet werden müssen. Zusätzlich kann die Signalisierung jedoch auch, wie im Abschnitt *Kooperation* beschrieben, über einfache Methodenaufrufe erfolgen. Jacks sind durch sog. *Communication Channels* verbunden. Diese können je nach Verbindungsart ausgetauscht werden: Befinden sich zwei Filter auf einem Knoten im Netzwerk, können die Daten über *Pointer Sharing* übertragen werden; befinden sie sich auf verschiedenen Knoten, wird ein Netzwerkprotokoll verwendet. Die Art des verwendeten Communication Channels ist für die beteiligten Filter transparent.

5. ZUSAMMENFASSUNG UND AUSBLICK

In diesem Paper wurden verschiedene Ansätze untersucht, um dynamische Komponentenframeworks für Multimediasysteme zu realisieren. Basierend auf dem Design Pattern *Pipes and Filters* wurden Erweiterungen vorgestellt, um die dynamische Komposition von Diensten aus Filtern und die verteilte Installation von Komponenten auf Servern im Netzwerk zu ermöglichen sowie die Signalisierung und den Datenfluss zwischen den Komponenten umzusetzen.

In den untersuchten und hier vorgestellten Arbeiten wird die Installation der Komponenten auf verschiedenen Servern stets als Voraussetzung für die Kombination der Komponenten zu Diensten angenommen. Da das Prinzip der Komponentenframeworks vorsieht, dass laufend neue Komponenten hinzugefügt werden können, würde es sich anbieten, diese basierend auf der Servicekonfiguration automatisch bei Bedarf auf den einzelnen Knoten zu installieren. Solche Verfahren für das automatische Deployment von Multimediadiensten wurden in dieser Arbeit nicht untersucht und eignen sich für weitere Untersuchungen.

Auch auf Sicherheitsaspekte wurde in diesem Paper nicht eingegangen. Anbieter von verteilten Multimedia-Komponentenplattformen möchten möglicherweise nur angemeldeten Nutzern gestatten, die auf ihren Servern installierten Filter zu nutzen. In diesem Fall müssen Vorkehrungen getroffen werden, um die Verarbeitung unauthorisierter Datenströme zu vermeiden.

Desweiteren wurde in dieser Ausarbeitung auf die Möglichkeit der automatischen Adaption der Servicekonfigurationen an die verfügbare Netzwerkbandbreite und sonstige Dienstgütekriterien hingewiesen. Hierfür ist eine ständige Überwachung der Netzwerkeigenschaften von Nöten, auf die hier nicht eingegangen wurde. Der interessierte Leser sei für die Vorstellung eines effizienten Verfahrens zur Bandbreitenüberwachung auf die Arbeit von Xu et al. [9] verwiesen.

Grundsätzlich ist zu erwarten, dass Multimedia-Komponentenframeworks aufgrund ihrer einfachen Erweiterbarkeit, der effizienten Wiederverwendbarkeit von Filtern und der daraus resultierenden beschleunigten Realisierbarkeit neuer Dienste sowie Adaptierbarkeit bestehender Dienste an neue Anforderungen eine immer wichtigere Rolle spielen werden. Schließlich ist es für Diensteanbieter von massiver Wichtigkeit, schnell auf neue Trends reagieren zu können und neuen Angeboten von Konkurrenten eigene entgegenzusetzen.

Insbesondere Systeme mit verteilten Komponenten können auch im Rahmen des derzeit massiv forcierten *Grid Computing* einen wichtigen Beitrag leisten, um rechenintensive Multimediaverarbeitung an externe Dienstleister auszulagern, ohne dabei große Veränderungen am Code der einzelnen Filter vornehmen zu müssen.

Schließlich können Komponentensysteme auch für Endbenutzer große Vorteile bringen, die auf einfache Weise aus vorgefertigten Komponenten eigene Multimedia-Services zusammenstellen und an ihre Bedürfnisse anpassen können. Hierzu müssen jedoch anwenderfreundliche Möglichkeiten entwickelt werden, solche Zusammenstellungen auf leicht verständliche Weise zu ermöglichen. Der Erfolg solcher Angebote hängt eindeutig von der Einfachheit der Benutzung ab.

6. LITERATURVERZEICHNIS

[1] Microsoft Corporation: *DirectShow*. http://msdn.microsoft.com/en-us/library/ ↩ ms783323.aspx - 12.11.2008.

[2] Apple Inc.: *QuickTime*. http://www.apple.com/quicktime/ - 12.11.2008.

[3] GStreamer Project: *GStreamer*. http://gstreamer.freedesktop.org/ - 12.11.2008.

[4] Xine Project: *Xine*. http://xinehq.de/ - 12.11.2008.

[5] Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal: *Pattern-orientierte Software-Architektur*, Kapitel Pipes-and-Filters, Seiten 54–71. Addison-Wesley, 2000.

[6] Bengel, Günther: *Grundkurs Verteilte Systeme: Grundlagen und Praxis des Client-server-computing*, Kapitel 3.2, Seiten 137–145. Vieweg+Teubner Verlag, 2004.

[7] Comer, Douglas: *Internetworking with TCP/IP: Principles, Protocols and Architecture*, Kapitel 21, Seiten 373–401. Prentice Hall, 2006.

[8] Wijnants, Maarten, Bart Cornelissen, Wim Lamotte und Bart De Vleeschauwer: *An overlay network providing application-aware multimedia services*. In: *AAA-IDEA '06: Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, New York, NY, USA, 2006. ACM.

[9] Xu, Dongyan und Xuxian Jiang: *Towards an integrated multimedia service hosting overlay*. In: *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, Seiten 96–103, New York, NY, USA, 2004. ACM.

[10] Wagner, Matthias und Wolfgang Kellerer: *Web services selection for distributed composition of multimedia content*. In: *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, Seiten 104–107, New York, NY, USA, 2004. ACM.

[11] Scholz, Andreas, Christian Buckl, Alfons Kemper, Alois Knoll, Jörg Heuer und Martin Winter: *WS-AMUSE - web service architecture for multimedia services*. In: *ICSE '08: Proceedings of the 30th international conference on Software engineering*, Seiten 703–712, New York, NY, USA, 2008. ACM.

[12] Lohse, Marco, Michael Repplinger und Philipp Slusallek: *An Open Middleware Architecture for Network-Integrated Multimedia*. In: *Protocols and Systems for Interactive Distributed Multimedia Systems, Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems, IDMS/PROMS 2002, Proceedings*, Band 2515 der Reihe *Lecture Notes in Computer Science*, Seiten 327–338. Springer, 2002.

[13] Jackson, M. und P. Zave: *Distributed feature composition: a virtual architecture for telecommunications services*. IEEE Transactions on Software Engineering, 24(10):831–847, Oct 1998.

[14] Cheung, E. und K.H. Purdy: *An Application Router for SIP Servlet Application Composition*. In: *IEEE International Conference on Communications, ICC '08*, Seiten 1802–1806, May 2008.

[15] Gibbs, Simon: *Object-Oriented Software Composition*, Kapitel Multimedia component frameworks, Seiten 305–319. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1995.

[16] Posnak, Edward J., R. Greg Lavender und Harrick M. Vin: *Adaptive Pipeline - an Object Structural Pattern for Adaptive Applications*. In: *Proceedings of the Third Pattern Languages of Programming Conference*, Monticello, Ill., Sept 1996.

[17] Object Management Group: *CORBA*. http://www.corba.org/ - 18.11.2008.

[18] Fuentes, Lidia und Josée M. Troya: *Towards an open multimedia service framework*. ACM Comput. Surv., 32(24):24–29, March 2000.