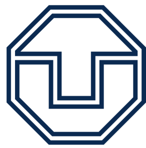
Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe. Wörtliche und sinngemäße Zitate sind als solche gekennzeichnet. Diese Arbeit wurde bisher weder in gleicher noch ähnlicher Form einer anderen Prüfungskommission vorgelegt und auch nicht veröffentlich.

Dresden, den 31.07.2007

Oliver Mroß



Großer Beleg

Entwurf und Umsetzung des Multimodality Services Component der EMODE Laufzeitumgebung

Bearbeitet von:	Oliver Mroß
Geboren am:	03.03.1983 in Hoyerswerda
Matr. Nr.:	2938033
Betreuender Hochschullehrer:	Prof. Dr. A. Schill
Betreuer:	Dipl.-Inf. Gerald Hübsch
Institut:	Systemarchitektur
Lehrstuhl:	Rechnernetze
Abgabe:	31.07.2007

Inhaltsverzeichnis

1. Einführung **5**
1.1. Aufbau und grundlegende Funktionen der EMODE Laufzeitumgebung 5
1.2. Ziel 7
1.3. Gliederung 7

2. Konzept und Stand der Technologien **8**
2.1. Multimodale Systeme 8
2.1.1. Konzeptuelle Struktur eines multimodalen Systems 8
2.1.2. Anforderungen an ein multimodales System nach EMODE 11
2.1.3. Eingabekomponenten 11
2.1.4. Ausgabekomponenten 13
2.2. Das Paradigma der Service-orientierten Architektur (SOA) 14
2.2.1. Begriffe 14
2.2.2. Schlüsselkonzepte der SOA 15
2.2.3. SOA im Vergleich mit anderen Paradigmen 16
2.2.4. Architekturmodell der EMODE Laufzeitumgebung 17
2.3. OSGi – Framework für Service-orientierte Anwendungen 19
2.3.1. Servicekomponenten – Bundles 20
2.3.2. Besondere Eigenschaften 20
2.4. Die EMODE-Anwendungen 21
2.4.1. Beschreibung des *User Interface* von EMODE Anwendungen 22

3. Entwurf **27**
3.1. Der Gesamtentwurf der MSC 27
3.1.1. Der Entwurf der MSC im Detail 28
3.1.2. Der Anwendungscontainer 30
3.2. Die Modalitäten 35
3.2.1. Die grundlegenden Funktionsweisen der Modalitäten 35
3.2.2. Die Beschreibung der Modalitäten durch Meta-Informationen 37
3.2.3. Interaktion durch Java Interfaces 38
3.2.4. Ein- und Ausgabe durch die Sprache 40
3.2.5. Direkte Ein- und visuelle Ausgabe 42
3.3. Der Entwurf der EMODE Anwendungen 43
3.3.1. Die Verarbeitung der Benutzereingaben durch die EMODE Anwendung . 45
3.3.2. Die Erzeugung der Ausgabedaten durch die EMODE Anwendung 45
3.3.3. Die Meta-Informationen der EMODE Anwendungen 46
3.4. Die Interpretation der Eingabe 51
3.4.1. Die Eingabeverarbeitung innerhalb der Eingabemodalität 52
3.4.2. Die Eingabeverarbeitung innerhalb des Anwendungscontainer 53
3.5. Adaption der Benutzerschnittstelle 55
3.5.1. Die Transformation der multimodalen Benutzerschnittstelle 56
3.5.2. Die Modifikation der Benutzerschnittstelle 56
3.6. Das dynamische Verhalten der MSC 58
3.6.1. Aktivieren einer Modalität 59

3.6.2. Deaktivieren einer Modalität . . . 59
3.6.3. Integration der EMODE Anwendungen . . . 60
3.6.4. Beenden einer EMODE Anwendung . . . 61

4. Implementierung **62**
4.1. Der Vergleich der Anwendungsanforderung mit den Eigenschaften der Modalität . 62
4.1.1. Der Vergleichsalgorithmus . . . 63
4.2. Der Verbindungsaufbau zwischen Modalität und EMODE Anwendung . . . 66
4.2.1. Das ModalityConnection-Objekt . . . 67
4.3. Die Analyse der Eingabedaten durch den Anwendungscontainer . . . 69
4.3.1. Die Verarbeitung der EMODE-Aktionen . . . 70
4.3.2. Die Verarbeitung der XForms-Aktionen . . . 72
4.4. Der Aufruf der Anwendungsfunktionen durch Kommandos und Reflektion . . . 74
4.4.1. Die Ermittlung der Kommandos und Argumente . . . 74
4.4.2. Vom Kommando zur Anwendungsfunktion . . . 75

5. Zusammenfassung **77**
5.1. Vergleich der EMODE MSC mit dem JANUS-System . . . 78
5.2. Die „*Proof-of-Concept*“ – EMODE Applikation . . . 80
5.2.1. Die Chat-Anwendung . . . 81
5.3. Ausblick . . . 83

A. UML Diagramme **85**
A.1. MSC Klassen . . . 86
A.2. Anwendungscontainer . . . 87
A.3. Modalitäten . . . 88
A.3.1. Aktivitäten der Spracheingabemodalität . . . 88
A.3.2. Entwurf der Modalität der direkten Manipulation . . . 89
A.4. Anwendungsfälle . . . 90
A.4.1. Überblick . . . 90
A.4.2. Aktivieren einer Modalität . . . 91
A.4.3. Deaktivieren einer Modalität . . . 92
A.4.4. Starten einer EMODE Anwendung . . . 93
A.4.5. Beenden einer EMODE Anwendung . . . 94
A.5. Methoden der Eingabeverarbeitung . . . 95
A.5.1. processMessage(...) Methode . . . 95
A.5.2. Ersetzen der EMODE-Aktionen . . . 96
A.5.3. Entfernen der XForms-Aktionen . . . 96
A.5.4. Verarbeitung der XForms-Aktionen . . . 97

B. Quellcode **98**
B.1. Vergleichsalgorithmus . . . 98
B.2. ModalityInput . . . 100
B.3. ModalityOutput . . . 101
B.4. Aufruf der Anwendungsfunktionen . . . 101
B.5. Schemadateien . . . 102
B.5.1. Kommando-Funktion-Bindung . . . 102

1. Einführung

Mit der zunehmenden Bedeutung von mobilen Endgeräten, wie bspw. Mobiltelefone oder PDA's, werden neue Bereiche in der Interaktion von Mensch und Maschine erschlossen. Die Fähigkeiten dieser Geräte, speziell die Ein- und Ausgabe von Informationen, sind begrenzt. Das Display eines Mobiltelefons ist relativ klein und kann daher weniger Informationen darstellen als gewöhnliche Computer. Weiterhin können auch Situationen auftreten, die die Bedienung des Gerätes erschweren bzw. unmöglich machen. Stellen Sie sich folgendes Szenario vor.

Ein Fahrradkurier erhält Transportaufträge via Handy. Dieses Handy kann durch die Betätigung der Tasten aktiviert werden. Während der Fahrt kann der Kurier aktuelle Benachrichtigungen durch Stimmenausgabe via Headset entgegennehmen. Er kann ebenfalls durch Stimmeneingabe Befehle an das Handy richten. Nun kann jedoch eine Situationen auftreten, in der die Stimmeneingabe ungeeignet ist. Während einer Fahrt nimmt die Umgebungslautstärke zu. Der Kurier kann kurzzeitig anhalten und die Transportaufträge mit Hilfe eines digitalen Stiftes durchblättern und bestätigen.

Der Mensch kann auf unterschiedlichen Wegen mit dem Endgerät interagieren. Die Informationen der Eingabe werden innerhalb eines bestimmten Anwendungskontext interpretiert. Dadurch werden weitere Verarbeitungsschritte ausgelöst. Innerhalb bzw. am Ende der Verarbeitung erzeugt das Gerät wiederum für den Menschen verständliche Ausgaben. Daraufhin kann der Mensch neue Eingaben durchführen und es entsteht somit ein Dialog zwischen dem Menschen und der Maschine. Die von der Maschine erzeugten Ausgaben können, wie die Eingaben des Menschen, unterschiedlicher Natur sein. Sie sind auf die Sinnesorgane des Menschen begrenzt.

Eine Möglichkeit der Eingabe von Informationen durch spezielle Signale (auditive Signale, Signale der direkten Manipulation von Eingabeelementen) werden durch den Begriff der Eingabemodalität bezeichnet. Analog dazu wird eine spezielle Ausgabemöglichkeit als Ausgabemodalität bezeichnet. Diese Begriffe werden im späteren Verlauf dieser Arbeit in verschiedenen Zusammenhängen genauer betrachtet und sind daher von fundamentaler Bedeutung. Werden mehrere Eingabemodalitäten, bzw. auch Ausgabemodalitäten, innerhalb einer Anwendung genutzt, so spricht man von einer multimodalen Anwendung.

In dieser Belegarbeit wird die Architektur und die Entwicklung eines Prototypen zur Realisierung der EMODE Laufzeitumgebung beschrieben. Sie soll die Entwicklung von multimodalen Anwendungen vereinfachen und deren Einsatz unabhängig von der jeweiligen Plattform ermöglichen. Dabei werden bereits bestehende Technologien genauer betrachtet und bewertet.

1.1. Aufbau und grundlegende Funktionen der EMODE Laufzeitumgebung

Die Laufzeitumgebung kann hauptsächlich in 3 Bereiche unterteilt werden. Folgende Bereiche können unterschieden werden:

1. die EMODE Anwendungen
2. die Komponente der multimodalen Dienste (Multimodality Services Component - MSC)

3. die Ein- und Ausgabegeräte

Eine Übersicht wird in der Abbildung 1.1 dargestellt.

Eine EMODE Anwendung ist eine eigenständige Applikation mit einem eigenen Kontext. Sie kann mehrere Eingabe- und Ausgabegeräte nutzen. Welche Geräte genutzt werden sollen, wird durch die Anwendung selbst definiert. Die Beschreibung wird an die MSC übergeben.

Die MSC bildet eine Vermittlungseinheit zwischen den EMODE Anwendungen und den Ein- und Ausgabegeräten. Sie stellt den EMODE Anwendungen bestimmte Dienste zur Verfügung, um die entsprechenden Ein- und Ausgabegeräte verwenden zu können. Die MSC verbindet geforderte Modalitäten mit der entsprechenden EMODE Anwendung. Dies soll so unabhängig wie möglich von der verwendeten Plattform geschehen, da besonders mobile Endgeräte stark in ihrer Beschaffenheit variieren können.

Die unterschiedlichen Eingabegeräte, z.B. Mikrophon, und die Ausgabegeräte, wie bspw. das Display, werden durch die Modalitäten repräsentiert. Analog zu den Ein- und Ausgabegeräten existieren Eingabe- und Ausgabemodalitäten. Sie sind die Komponenten, die auf der Ebene der Software den Zugriff auf die Ein- und Ausgabegeräte durchführen. Des Weiteren werden sie durch die MSC den EMODE Anwendungen als Service zur Verfügung gestellt. Daraus folgt, dass unterschiedliche Modalitäten zu gleichen Zeit verwendet werden können. Der parallele Ablauf der Komponenten soll durch die Laufzeitumgebung gewährleistet werden.

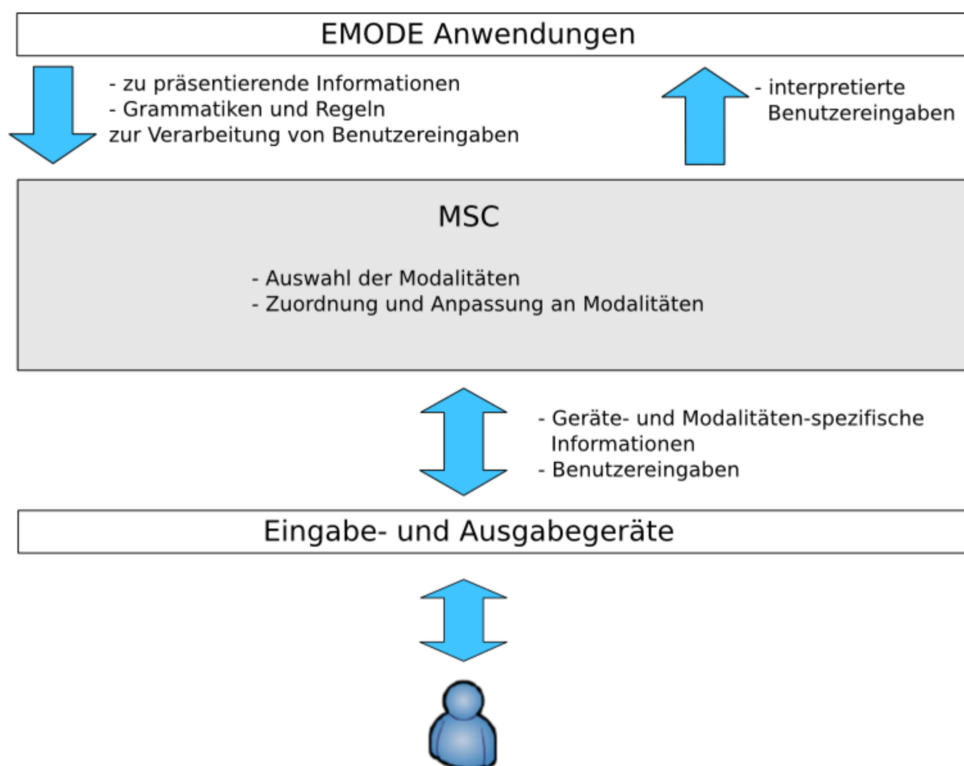


Abbildung 1.1.: EMODE Kernbereiche [HHN]

Eine weitere Aufgabe der EMODE Umgebung ist die Verwaltung der Modalitäten in den unterschiedlichsten Situationen, wie im vorherigen Szenario beschrieben. Dabei ist es notwendig, die Modalitäten dynamisch zur Laufzeit einbinden und aktivieren bzw. deaktivieren zu können. Das heißt, die Verfügbarkeit der Modalitäten kann sich im Verlauf einer Anwendung verändern. Informationen können durch alternative Modalitäten ausgegeben werden, welche in entsprechenden Situationen besser geeignet sind. All das muss die MSC verarbeiten können.

Wie zuvor erwähnt, soll EMODE soweit wie möglich plattformunabhängig sein. Das heißt, es ist notwendig, dass die Kommunikation der Modalitäten mit den EMODE Anwendungen ebenfalls plattformunabhängig durchgeführt wird. Dementsprechend müssen Zwischenformate in der Kommunikation zwischen den Anwendungen und den Komponenten eingesetzt werden. Speziell die Eingabe des Nutzers muss unabhängig vom Eingabegerät interpretiert und im Zwischenformat an die Anwendung übergeben werden.

Eine weitere Anforderung, die sich aus der Plattformunabhängigkeit und der Unterstützung mehrerer Modalitäten zu gleichen Zeit ergibt, ist die abstrakte Beschreibung der Benutzerschnittstelle einer EMODE Anwendung. Die unterschiedlichen Ein- und Ausgabegeräte besitzen, in Bezug auf ihre Eignung zur Darstellung von Informationen, unterschiedliche Eigenschaften. Bei der auditiven Ausgabe von Informationen werden die Elemente des *User Interface*, im Vergleich zur visuellen Ausgabe, verschieden präsentiert. Die Ausgabe von Informationen erfolgt zeitlich sequentiell und kann bei Nachfrage wiederholt werden, hingegen werden visuelle Informationen in räumlicher Ordnung zueinander ausgegeben. Daher ist es wichtig, dass die Benutzerschnittstelle der Anwendung soweit wie möglich unabhängig von den Ein- und Ausgabegeräten beschrieben wird. Diese Beschreibung dient der MSC als Ausgangspunkt für die Anpassung der Benutzerschnittstelle, an die zu verwendende Modalität und dessen Ausgabegerät.

1.2. Ziel

Das Ziel der Arbeit ist die Entwicklung der EMODE Laufzeitumgebung. Dabei sollen bestehende Technologien auf die mögliche Integration, in die Laufzeitumgebung, bewertet werden. Des Weiteren soll als sogenanntes „*Proof–of–Concept*“ ein Einsatzszenario der EMODE Laufzeitumgebung geschaffen werden. Eine *Chat*–Anwendung soll durch unterschiedliche Ein- und Ausgabemodalitäten gesteuert werden können. Die Eingabe von Daten soll mittels der Tastatur, Maus und Stimme erfolgen. Die Anwendung soll Informationen visuell und auditiv wiedergeben können.

Mittels des „*Proof–of–Concept*“ soll das Zusammenspiel der unterschiedlichen Technologien untersucht werden. Wichtig dabei ist der Aspekt der Plattformunabhängigkeit von EMODE, um die zukünftige Unterstützung mobiler Endgeräte gewährleisten zu können.

1.3. Gliederung

Das 2. Kapitel dieser Arbeit stellt die bisherigen Konzepte und Technologien vor, die als Grundlage für die Entwicklung der EMODE Laufzeitumgebung dienen sollen. In diesem Kapitel werden die verschiedenen Paradigmen miteinander verglichen und anhand der Anforderungen der EMODE Spezifikation nach [HHN] bewertet.

Ausgehend von den in Kapitel 2 beschriebenen Konzepten und Technologien, wird im 3. Kapitel der Entwurf der EMODE Laufzeitumgebung erläutert, dabei liegt der Schwerpunkt auf dem softwaretechnische Entwurf. Des Weiteren werden eigene Ideen und Konzepte vorgestellt und in den Entwurf der Laufzeitumgebung integriert.

Kapitel 4 präsentiert die Implementierung wichtiger Funktionen anhand von Quellcodeauschnitten. Dabei soll ein Einblick in relevante funktionale Aspekte ermöglicht werden.

Im letzten Kapitel werden bestimmte Entwurfsentscheidungen zusammenfassend beschrieben und im Vergleich mit dem Janus-System charakterisiert. Im Anschluss wird der Prototyp des „*Proof-of-Concept*“ präsentiert und nachfolgend mögliche Erweiterungen vorgestellt.

2. Konzept und Stand der Technologien

In diesem Abschnitt werden die zugrunde liegenden Konzepte beschrieben und bereits existierende Technologien in Bezug auf die Realisierung der Konzepte bewertet. Als erstes wird der Begriff des multimodalen Systems erläutert und beschrieben, welche Anforderungen diese erfüllen müssen. Danach wird untersucht welche Technologie am besten für die Realisierung der Architektur der EMODE Laufzeitumgebung geeignet ist.

2.1. Multimodale Systeme

Die Kommunikation von Mensch zu Mensch kennzeichnet sich dadurch aus, dass neben der Sprache, auch andere Elemente des menschlichen Verhaltens, wie Gestiken, Blickkontakte und Gesichtsausdrücke teilweise bewusst, teilweise unbewusst wahr genommen werden. Das Zusammenspiel dieser Elemente ermöglicht eine bessere Wahrnehmung und ein besseres Verständnis für die ausgetauschten Informationen. Mit Hilfe eines multimodalen Systems soll die menschliche Kommunikation nachgebildet werden.

Nach dem Paper von Laurence Nigay und Joelle Coutaz [NC93] wird ein multimodales System wie folgt definiert. Im Allgemeinen unterstützt ein multimodales System die Kommunikation mit dem Nutzer durch mehrere Modalitäten (Stimme, Gestiken, Tastatur, etc.). Der Begriff der Modalität bezieht sich auf den Typ des Kommunikationskanals, um Informationen zu übermitteln oder zu erfassen.

Eine Modalität bestimmt den Typ der auszutauschenden Daten, z.B. akustische Daten. Zusätzlich muss auch der gegenwärtige Kontext des Systems betrachtet werden, in dem die Daten eine Bedeutung erhalten. Dies unterscheidet ein multimodales System von einem multimedialen System. Dieses kann ebenfalls die Informationen auf den verschiedenen Kommunikationskanälen übermitteln. Verschiedene Geräusche oder die Stimme des Nutzers werden durch Lautsprecher ausgegeben. Graphiken und Texte werden durch das Display dargestellt.

Ein multimodales System verarbeitet die auszutauschenden Informationen auf einer höheren Abstraktionsebene, d. h. die Semantik der übermittelten Daten wird betrachtet. Die Eingabemodalitäten können ausgetauscht bzw. miteinander kombiniert werden. Dies bietet den Vorteil, das Nutzer verschiedene Eingaben simultan durchführen können. Zum Beispiel kann mit Hilfe eines Eingabestiftes (Stylus) ein Teil eines Textes markiert und danach durch ein Sprachkommando entfernt werden. Dazu muss die Information der einen Modalität, der ausgewählte Textbereich, mit dem Löschkommando der Spracheingabemodalität verknüpft werden. Dadurch kann die Kommunikation mit dem Computer in einem, für den Menschen, mehr natürlicheren Weg als bisher erfolgen.

2.1.1. Konzeptuelle Struktur eines multimodalen Systems

Im Entwurf des multimodalen Systems soll nach [DA05] die interne Darstellung der Eingabedaten des Nutzers, die Funktionen des Systems und die Interaktionsmodalitäten unabhängig

voneinander gestaltet werden. Das Ziel dabei ist, dass die Eingaben des Nutzers in eine geräteunabhängige Darstellung konvertiert werden. Somit macht es für das System keinen Unterschied, woher die Eingabedaten kommen, z.B. kann der Nutzer entweder per Stimme das Kommando „Abbrechen“ auslösen oder die ESC-Taste des Keyboards betätigen. Die Unabhängigkeit soll durch eine einheitliche Schnittstelle gewährleistet werden.

Die Abbildung 2.1 stellt die konzeptuelle Struktur des multimodalen Systems schematisch dar. Die einzelnen Konzeptelemente werden durch die nachfolgenden Punkte anhand der Abbildung erläutert.

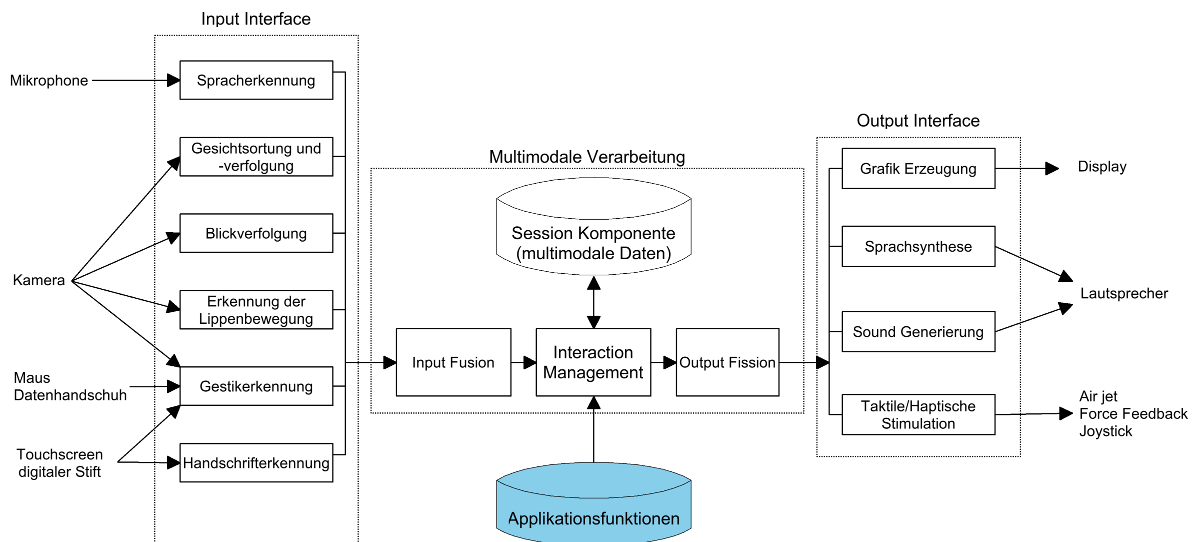


Abbildung 2.1.: konzeptuelle Struktur eines multimodalen Systems [DA05]

Input Interface

Heutige Computer besitzen neben dem Keyboard und der Maus zusätzlich weitere Eingabegeräte, z. B. Mikrophon, Videokamera, Scanner, Joysticks usw.. Diese Geräte variieren in ihrer Funktionsweise und der Art der Eingabedaten. Die Maus erzeugt Daten, bezüglich der Position des Cursors. Die Videokamera wandelt die Bilder des Benutzers in eine Datenrepräsentation um. Die Bilddaten können wiederum für die Erkennung von Gestiken, Lippenbewegungen und Gesichtern verwendet werden. Das Mikrophon liefert Daten der Spracheingabe des Benutzers, die in der automatischen Spracherkennung benötigt werden.

Das *Input Interface* bietet eine allgemeine Schnittstelle für die Eingabemodalitäten und den dazugehörigen Eingabegeräten.

Output Interface

Die Ausgaben eines multimodalen Systems können, genauso wie die Eingaben, unterschiedlich ausgeprägt sein. Um auch hier die unterschiedlichen Eigenschaften der Ausgabegeräte überbrücken zu können, muss ein multimodales System eine Schnittstelle für den Zugriff auf die Ausgabemodalitäten und den dazugehörigen Ausgabegeräten anbieten.

Interaction Management

Die Koordinierung und Überwachung der Datenflüsse der Ein- und Ausgabeinformationen, wird durch das *Interaction Management* vorgenommen. Eine weitere Aufgabe der Komponente ist die Bewahrung des Anwendungskontext, d. h. zugehörige Ressourcendateien müssen in den Ablauf des multimodalen Systems integriert werden. Der gegenwärtige Zustand der Interaktion muss zusätzlich gesichert werden. Dies ist notwendig, um Veränderungen im System handhaben zu können, bspw. wenn eine neue Eingabemodalität verfügbar wird.

Session Komponente

Zur Unterstützung des Interaktionsmanagement führt die Komponente die Sicherung der Eingabedaten, bzw. der Ausgabedaten, innerhalb einer Sitzung durch. Die Daten können dabei, je nach Anwendung, persistent oder temporär gesichert werden. Als Beispiel dient hier die Nachrichten-Historie einer *Chat*-Anwendung, die persistent und nutzerabhängig gesichert wird.

Eine weitere wichtige Funktion ist die Replikation der Zustände der verschiedenen Modalitäten. Das folgende Beispiel soll dies verdeutlichen. Eine Anwendung besitzt eine graphische Benutzeroberfläche (GUI) und eine Spracheingabeschnittstelle (SUI). Der Benutzer kann zwischen beiden Schnittstellen wählen bzw. ein Teil der Eingabe durch die Eingabeelemente der GUI durchführen und den anderen Teil über die SUI vervollständigen. Damit im nächsten Schritt die GUI alle Eingabedaten anzeigen kann, müssen alle Modalitäten in einem synchronen Zustand gehalten werden.

Der Mechanismus der Zustandsreplikation ist ebenfalls Voraussetzung für das Auswechseln von Modalitäten in den verschiedenen Situationen. Wird eine neue Modalität in das multimodale System hinzugefügt, so müssen alle vorherigen Eingabe- bzw. Ausgabedaten an die neue Modalität übergeben werden, um den aktuellen Interaktionszustand zu gewährleisten.

Input Fusion

Die Informationseinheiten, die die verschiedenen Eingabemodalitäten bereitstellen, werden durch die Komponente miteinander verknüpft. Die Vereinigung der Informationen findet nach [HHN] auf der Signalebene und auf semantischer Ebene statt. Die Kombination auf Signalebene soll die zeitlichen Beziehungen der erzeugten Signale sicherstellen und ist Grundlage für die Vereinigung der Informationen auf semantischer Ebene. Sie soll alle Eingabeinformationen in ihrer Gesamtheit erfassen und für die weitere Verarbeitung aufbereiten. Genauere Informationen zur Vereinigung der Daten können in [HHN] und [DA05] nachgeschlagen werden. Die Entwicklung der Komponente zur Vereinigung der Eingabedaten wird in dieser Arbeit nicht betrachtet.

Output Fission

Die Komponente führt die Trennung der Ausgabeinformationen in bestimmte Ausgabeteile durch. Diese werden auf die entsprechenden Ausgabemodalitäten verteilt und anhand von zeitlichen Richtlinien koordiniert. Genauere Informationen dazu können ebenfalls in [HHN] und [DA05] nachgelesen werden.

Applikationsfunktionen

Diese Komponente stellt die Menge aller möglichen Funktionen der Applikation bereit. Aus dieser Menge und anhand der Eingaben des Nutzers entscheidet die Komponente des *Interaction Management* welche Funktionen aufgerufen werden soll.

2.1.2. Anforderungen an ein multimodales System nach EMODE

Nachdem die konzeptuelle Struktur eines mulitmodalen Systems erläutert wurde, muss geklärt werden, welche Anforderungen die Komponente der multimodalen Dienste (MSC der EMODE Laufzeitumgebung) besitzt. In der EMODE Spezifikation [HHN] werden folgende Bedingungen an die MSC gestellt.

1. Die Verfügbarkeit der Modalitäten kann sich im Verlauf der Anwendung verändern.
2. Die MSC soll offen und flexibel sein, um neue Modalitäten ohne großen Aufwand unterstützen zu können.
3. Verschiedene Implementationen einer Modalität können mit unterschiedlichen Systemanforderungen und in unterschiedlichen Qualitäten existieren.
4. Die Dienste der Modalitäten können verteilt sein, daher soll jede Modalität als eigenständiger Service modelliert werden.

Diese Bedingungen sollen nach der EMODE Spezifikation [HHN] durch das Paradigma der Service-orientierten Architektur erfüllt werden. Im Abschnitt 2.2 wird dieses Paradigma näher erläutert.

Die MSC der EMODE Laufzeitumgebung besitzt die Funktionen der Ein- und Ausgabeverarbeitung. Die Konzepte, die hinter den Verarbeitungsprozessen stehen, sollen in den folgenden Abschnitten 2.1.3 und 2.1.4 erläutert werden.

2.1.3. Eingabekomponenten

Wie in der Abbildung 2.1 dargestellt, integriert ein multimodales System verschiedene Eingabegeräte in den Ablauf einer Anwendung. Dabei müssen die unterschiedlichen Eingabedaten verarbeitet werden können. Die Unterschiede der Eingabegeräte und der von ihnen erzeugten Daten können auf verschiedenen Wegen überbrückt werden.

Eine Möglichkeit ist, eine begrenzte Menge von Eingabegeräten in die Anwendung selbst fest zu integrieren. Die Anwendung kann für die festgelegten Eingabegeräte spezielle Funktionen bereitstellen und somit diese Geräte noch besser unterstützen. Die Daten der Eingabegeräte können ebenfalls durch spezielle Funktionen direkt verarbeitet werden. Dadurch kann der gesamte Ablauf der Anwendung schneller von statten gehen.

Der große Nachteil des Entwurfs ist folgender. Die Anwendung kann nur durch die festgelegten Eingabegeräte bedient werden, d. h. die Erweiterung der Anwendung durch neue Eingabegräte ist ohne Umstände nicht möglich. Somit ist die Anforderung nach Flexibilität und Offenheit [HHN] nicht erfüllt.

Ein weiterer Entwurf für die Überbrückung der Unterschiede der Eingabedaten ist die Konvertierung in ein einheitliches Zwischenformat. Die zu den Eingabegeräten zugehörigen Eingabemodalitäten müssen sich um die Konvertierung der Daten in das Zwischenformat kümmern. Dieser Prozess wird als Interpretation der Eingabedaten bezeichnet. Die folgende Abbildung 2.2 soll die

Umsetzung der Eingabedaten in das Zwischenformat verdeutlichen, dabei dienen die nachfolgenden Punkte der Erläuterung der dargestellten Komponenten in Abb. 2.2.

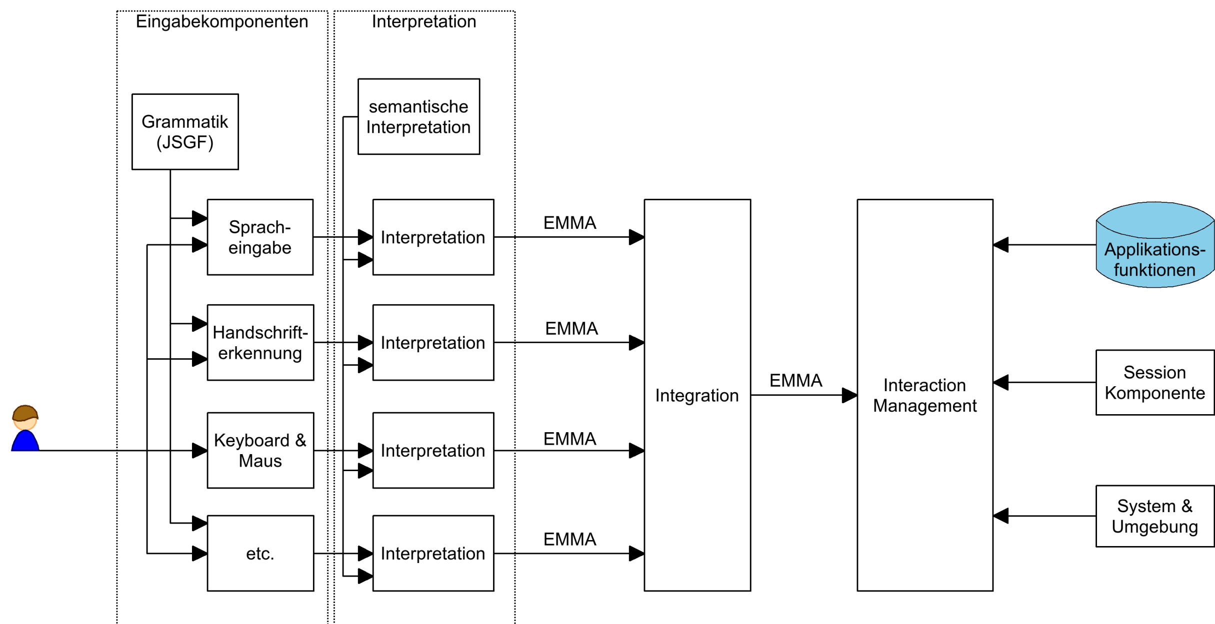


Abbildung 2.2.: Eingabekomponenten des multimodalen Systems

Interpretation

Jede Eingabemodalität muss eine semantische Interpretation der Eingabedaten durchführen. In diesem Prozess wird die Bedeutung der Eingabedaten, abhängig von der jeweiligen Modalität ermittelt und in das EMMA Austauschformat überführt. Dabei wendet die jeweilige Modalität spezifische Transformationsregeln an. Zum Beispiel können die Wörter einer Spracheingabemodalität, wie „senden“ , „abschicken“ oder „übermitteln“ als „sende“ Kommando dargestellt werden. Ein Klick auf einen „Sende“ -Button wird als das gleiche Kommando interpretiert.

Integration

Diese Komponente beinhaltet die Vereinigung der Eingabedaten, wie sie bereits durch die Abbildung 2.1, durch die *Input Fusion*-Komponente, beschrieben wird.

System & Umgebung

Damit das Interaktionsmanagement fähig ist, auf Veränderungen im System reagieren zu können, benötigt es eine Komponente, die diese erkennt. Die Veränderungen können im Bereich der Fähigkeiten oder der Verfügbarkeit der Ein- und Ausgabegeräte liegen. Das System muss erkennen können, ob der Benutzer z. B. die Lautsprecher stumm geschalten hat oder nicht.

2.1.3.1. Die Anforderungen an das Zwischenformat der Eingabeinformationen

Das Zwischenformat, welches als Informationsträger zwischen den Eingabemodalitäten und der MSC dienen soll, muss folgende Anforderungen erfüllen:

- Es muss eine plattform- und geräteunabhängige Eigenschaft aufweisen, um die Anforderungen nach [HHN] zu erfüllen.
- Um eine breite Unterstützung durch die Eingabegeräte zu ermöglichen, muss es eine Teilmenge des XML-Standards sein.
- Damit die verschiedenen Eigenschaftsdaten der Eingabegeräte an die MSC übergeben werden können, muss das Zwischenformat flexibel und erweiterbar sein.

In [LRR+03] und in [HHN] wird das *Extensible Multimodal Markup Annotation* (EMMA) Format als Zwischenformat der interpretierten Eingabedaten vorgeschlagen. Es besitzt folgende Eigenschaften:

- EMMA ist eine Teilmenge des XML-Standards und erfüllt somit die Anforderung nach Geräte- und Plattformunabhängigkeit.
- Das EMMA-Element `<emma:info>` bietet die Möglichkeit zusätzliche Informationen von Geräteherstellern oder Anwendungen aufzunehmen. Damit ist das Format flexibel und erweiterbar [CDJ+04].

2.1.4. Ausgabekomponenten

Auch die Ausgabegeräte besitzen, in Bezug auf ihre Funktionsweise und der Art der Ausgabedaten, unterschiedliche Eigenschaften. Um diese zu überbrücken, existieren, wie bei den Eingabegeräten, mehrere Möglichkeiten. Es kann ebenfalls eine begrenzte Menge von Ausgabegeräten statisch in eine Anwendung integriert werden. Auch hier besteht der Nachteil, dass neue Ausgabegeräte nicht in die Anwendung eingebunden werden können.

Analog zu den Eingabegeräten muss ein Zwischenformat für die Ausgabe der Daten verwendet werden. Dieses Zwischenformat fungiert als allgemeine multimodale Beschreibung des *User Interface* (MMUID). Die Ausgabeinformationen, die in der MMUID enthalten sind, müssen an das jeweilige Ausgabegerät angepasst werden. Die Anpassung des MMUID an das Ausgabegerät wird als Adaption bezeichnet. Die folgende Abbildung soll dies verdeutlichen. Die Komponenten, die in der Abbildung 2.3 dargestellt sind, werden durch die nachfolgenden Punkte erläutert.

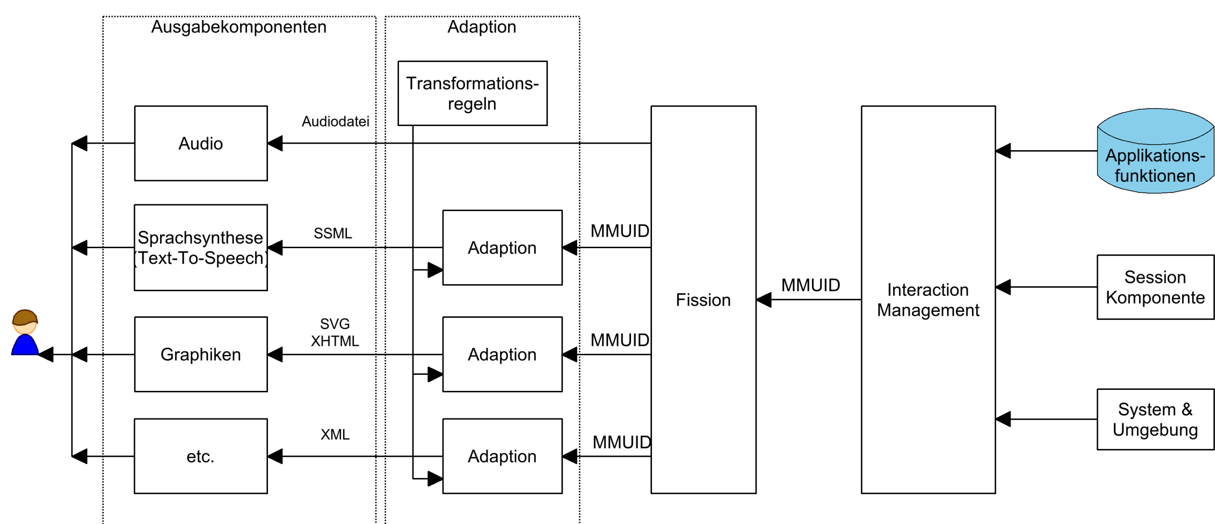


Abbildung 2.3.: Ausgabekomponenten des multimodalen Systems

Fission

Die Komponente ermittelt, welche Ausgabemodalitäten für die Informationsdarstellung genutzt werden sollen und führt die Trennung der Ausgabeinformationen durch. Die MMUID kann für bestimmte Ausgabemodi, z. B. die visuelle Darstellung der Informationen, spezifische Anweisungen enthalten. Diese können Layoutanweisungen sein oder spezielle Informationen für das entsprechende Ausgabegerät, z. B. die Lautstärke der Lautsprecher.

Die Ausgabekomponenten können sich gegenseitig ergänzen, bzw. während der Anwendung zusätzliche Informationen bereitstellen, z. B. können graphische Elemente der GUI durch Sprachausgabeinformationen näher beschrieben werden. Spezielle Ausgabedateien, bspw. Audiodateien, können durch die Komponente direkt an die Ausgabemodalitäten übergeben werden.

Adaption

Um die allgemeine Beschreibung der Benutzerschnittstelle an die entsprechenden Ausgabegeräte anzupassen, muss diese unter Umständen transformiert werden. Die Transformation des MMUID wird anhand von speziellen Regeln durchgeführt. Die Ausgabemodalität muss diese bereitstellen. Eine Möglichkeit der Transformation von Daten bietet der XML-Standard in Form der XSLT-Technologie. Dabei muss das MMUID selbst im XML-Format vorliegen. Auf die Beschreibung des multimodalen *User Interface*'s wird im Abschnitt 2.4.1.2 eingegangen.

Der nachfolgende Abschnitt erläutert die Grundlagen der Service-orientierten Architektur. Sie stellt einen Weg dar, die Komponenten des multimodalen Systems im Aspekt einer Systemarchitektur zu betrachten.

2.2. Das Paradigma der Service-orientierten Architektur (SOA)

Im realen Leben kann ein Dienst allein oder in Kombination mit weiteren Diensten die Anforderungen von Personen oder Organisationen erfüllen. Im Bereich der verteilten Anwendungen können Dienstnutzer nicht nur Personen, sondern auch entfernte Rechner bzw. eigenständige Applikationen sein.

Das Paradigma der Service-orientierten Architektur beinhaltet die Organisation und die Verwendung verteilter Dienste.

Im Referenz Modell der SOA [MLM+06] wird ein Service folgendermaßen definiert.

„Ein Service besteht aus den Mechanismen, die auf der einen Seite die Bedürfnisse, und auf der anderen Seite mögliche Fähigkeiten miteinander verbinden.“

2.2.1. Begriffe

Die folgenden Begriffe werden im Verlauf der Arbeit verwendet und dienen der Beschreibung weiterer Konzepte.

Service Provider Im Allgemeinen sind es Personen oder Organisationen die Kompetenzen als Dienstleister besitzen. Im Bereich der verteilten Anwendungen bezieht sich der Begriff ebenfalls auf Applikationsinstanzen, die bestimmte Funktionalitäten gegenüber anderen Anwendungen anbieten.

Service Requester Eine Instanz die angebotene Dienste nutzen möchte. Genauso wie der *Service Provider*, kann der *Service Requester* im Bereich der verteilten Anwendungen eine eigenständige Applikation sein.

Service Registry Eine Registrierung, die die Beschreibungen und die Referenzen der verfügbaren Dienste enthält. Durch sie können die *Service Requester* unbekannte Dienste auffinden. Die *Service Registry* wird durch den *Service Provider* zur Verfügung gestellt.

2.2.2. Schlüsselkonzepte der SOA

Die Schlüsselkonzepte bilden die Grundlage für das Verständnis der Entwicklung neuer Konzepte, die besonders im Entwurf der Modalitäten ausschlaggebend sind.

Sichtbarkeit

Die angebotenen Dienste werden durch Servicebeschreibungen charakterisiert. Diese enthalten Informationen zu funktionalen Aspekten, technischen Anforderungen und Zugriffsmechanismen. Mit Hilfe der Informationen kann der *Service Requester* entscheiden, ob er den angebotenen Dienst nutzen möchte oder nicht. Folgende Informationselemente muss der *Service Requester* besitzen.

Informationen über:

1. die Existenz und Verfügbarkeit eines Dienstes
2. die Menge der Funktionen, die durch den Dienst erbracht wird
3. die Menge der Bedingungen und Regeln, unter denen der Dienst eingesetzt werden kann
4. die Menge der Regeln, unter denen der Dienst, durch den *Service Requester*, eingesetzt werden soll
5. die Art und Weise wie mit dem Dienst interagiert werden soll, um die gewünschten Ziele zu erreichen
6. das Format und den Inhalt der Informationen, die zwischen dem Service und dem *Service Requester* ausgetauscht werden sollen

Interaktion

Die Dienste werden durch den Austausch von Informationen und durch den Aufruf von Aktionen genutzt. Dies geschieht auf Basis des sog. Informationsmodell. Es charakterisiert die Informationen und Daten. Dabei wird das Format der Informationen, strukturelle Beziehungen innerhalb der Informationen, sowie spezifische Protokolle und Kommandos die Aktionen auslösen, definiert. Es bildet die Schnittstelle (*Service Interface*) zum jeweiligen Dienst und ist ein fundamentaler Bestandteil des SOA Paradigma. [MLM+06] Die Interaktion kann vielseitig ausgeprägt sein, findet jedoch hauptsächlich im Ausführungskontext des *Service Requester* statt.

Service Effekt

Der *Service Requester* möchte, durch den Einsatz eines Dienstes, ein bestimmtes Ergebnis realisieren. Anhand der Vorgabe eines zu erreichenden Resultates (der sog. *Real-World-Effect* [MLM+06]) führt der *Service Requester* die Auswahl der zu nutzenden Dienste durch. Zum Beispiel soll das Nachrichtenfenster einer *Chat*-Anwendung sprachlich ausgegeben werden. Der Anwendung steht der TTS-Service (*Text-To-Speech*) zu Verfügung. Im nächsten Schritt sucht die Anwendung nach dem Dienst, der die Vorgabe erfüllt. Der TTS-Service wird anhand seiner Beschreibung erfolgreich erkannt und kann durch die Anwendung genutzt werden. Das Ergebnis ist in diesem Fall die sprachliche Ausgabe der darzustellenden Informationen. Das Beispiel zeigt, dass der Effekt eines Dienstes, also das zu realisierende Ergebnis, vor der Nutzung des Service bekannt sein muss.

2.2.3. SOA im Vergleich mit anderen Paradigmen

Der Fokus der SOA liegt im Erreichen eines Ergebnisses, durch die Nutzung eines oder mehrerer Dienste. Hingegen liegt der Fokus der Objektorientierung (OO) in der Vereinigung von Operationen und Daten innerhalb eines Objektes.

Unterschiede der Paradigmen:

- In der OO können Methoden als Bestandteil eines Objektes angesehen werden.
- Innerhalb der SOA werden die Dienste als Zugang zu den Methoden betrachtet, dabei ist die Existenz des zugehörigen Objektes nebensächlich.
- Um ein Objekt nutzen zu können, muss es in der OO instantiiert werden, während ein Service nur dann genutzt werden kann wenn er bereits existiert.
- Objekte besitzen nach dem Paradigma der OO eine genaue Struktur, die jedoch mit keiner Semantik behaftet ist. Es kann nicht erkannt werden, welche Methode zu welchem Ergebnis führt. Im Gegensatz hebt die SOA die Auswirkung des Einsatz einer Methode besonders hervor. Für den *Service Requester* ist es wichtig zu wissen, welcher Dienst genutzt werden kann.

2.2.3.1. Vorteile der SOA

Die Vorteile der SOA gegenüber anderen Paradigmen, wie z. B. das *Client/Server*-Paradigma, sind folgende.

Durch das „*Blackbox-Modell*“ der Dienste und deren plattformunabhängige Beschreibungen können Systeme nach dem Paradigma der SOA in heterogenen Umgebungen eingesetzt werden. Dies bedeutet, dass der *Service Provider, der Service Requester* und die Dienstkomponenten auf unterschiedlichen Technologien basieren und trotzdem miteinander interagieren können.

Aufgrund der *Service Registry* können Dienstkomponenten zu beliebiger Zeit hinzugefügt bzw. auch entfernt werden. Der *Service Requester* muss sich danach erneut über die verfügbaren Dienste informieren bzw. durch den *Service Provider* informiert werden. Die Strategien können dabei vielfältig sein und sollen in dieser Arbeit nicht weiter betrachtet werden.

Die SOA besitzt gegenüber dem klassischen *Client/Server*-Modell dynamisch, flexible Eigenschaften. Im *Client/Server*-Modell werden die Dienste, die durch den Server angeboten werden nicht explizit beschrieben. Somit können die Dienstnutzer nicht, zur Laufzeit vorhandene Dienste, bewerten und dynamisch auswählen. Dies ist jedoch für die Umsetzung der MSC der EMODE Laufzeitumgebung notwendig. Der Kontext einer EMODE Anwendung kann sich im Verlauf der

Ausführung ständig verändern, z. B. unterschiedliche Umgebungslautstärken. Dies schlägt sich in der Verfügbarkeit und der Einsatzfähigkeit der Dienstkomponenten nieder, dementsprechend müssen die Dienste zur Laufzeit aktiviert bzw. deaktiviert werden können.

2.2.4. Die Anwendung des SOA Konzept im Architekturmodell der EMODE Laufzeitumgebung

Folgende Abbildung 2.4 zeigt das Modell der EMODE Laufzeitumgebung nach dem Prinzip der Service-orientierten Architektur mit Integration des *Frameworks* der *Open Service Gateway Initiative* (OSGi).

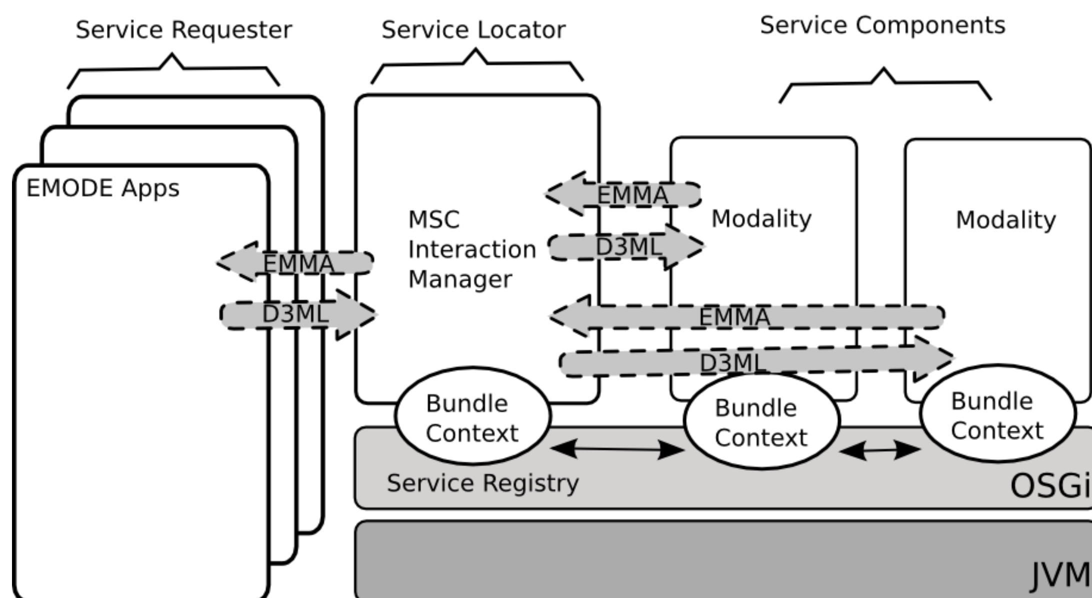


Abbildung 2.4.: EMODE Architektur

Das *OSGi-Framework* bietet, nach dem *Technical-WhitePaper* [OSG05] der *OSGi Alliance*, ein leicht gewichtiges Dienstmodell zum Veröffentlichen, Aufsuchen und Binden von Services innerhalb einer Virtuellen Maschine (Java VM) an. Es ermöglicht ein flexiblen Anwendungsentwurf auf Basis der SOA. Die Eigenschaften und technischen Details zum *OSGi Framework* werden in [OSG06] erläutert. Auf die Besonderheiten des *Frameworks* wird im folgenden Abschnitt 2.3 dieser Arbeit eingegangen, zunächst wird das Konzept der EMODE Architektur auf Basis der SOA erläutert.

Überträgt man das Konzept der SOA auf das Modell der multimodalen Ein- und Ausgabe, so stellen die einzelnen Modalitäten Dienstkomponenten dar. Sie realisieren die Ein- bzw. Ausgabe von Informationen als Dienste, sog. *Service Components* (siehe Abb. 2.4), welche durch eine EMODE Applikation genutzt werden können. Die Art und Weise der Implementierung der einzelnen Modalitäten spielt dabei keine Rolle. Wichtig ist es zu wissen welche Information durch welche Modalität angeboten wird bzw. welche Modalität welche Information wie ausgibt.

In Abbildung 2.5 werden die Beziehung der einzelnen Rollen der Komponenten der SOA verdeutlicht.

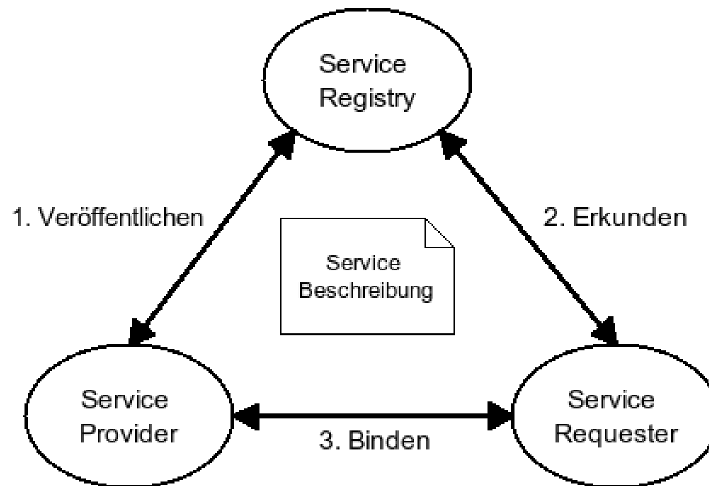


Abbildung 2.5.: Zusammenhänge der SOA [Emb04]

Damit eine Modalität, von außen durch eine EMODE Anwendung, angesprochen werden kann, muss der Dienst, den diese Modalität enthält, veröffentlicht werden. Die Service-orientierte Architektur bietet dazu eine Dienstregistrierung an, die *Service Registry*. Diese wird durch die MSC in der Rolle des *Service Provider* zur Verfügung gestellt. Die EMODE Anwendung nimmt die Rolle des *Service Requester* an. Mit Hilfe der *Service Registry* kann der *Service Requester* die nötigen Dienste aufsuchen. Nach diesem Schema ermittelt eine EMODE Anwendung, durch die MSC, die gewünschten Ein- und Ausgabedienste (Modalitäten). Wollen mehrere EMODE Anwendungen zur Laufzeit ein und den selben Dienst nutzen, so muss beachtet werden, dass die Ein- bzw. Ausgabegeräte in vielen Fällen immer nur durch eine Anwendung genutzt werden können. Zum Beispiel ist die parallele Ausgabe von Informationen durch akustische Signale ungeeignet, denn es können dabei Interferenzeffekte auftreten. Die Informationen werden dabei nur erschwert bzw. gar nicht durch den Menschen wahr genommen.

Die Dienste werden durch die Servicebeschreibungen charakterisiert. Die Beschreibungen enthalten Angaben über die Modalität, z. B. der Typ der Modalität oder genauere Informationen zum Ein- oder Ausgabegerät. Dadurch kann die MSC die jeweilige Modalität identifizieren und die Anfragen der verschiedenen EMODE Anwendungen entsprechend der Anforderungen an Ein- bzw. Ausgabedienste an die Modalitäten weiterleiten.

Durch die Kapselung der Modalitäten nach dem „*Blackbox-Modell*“ , ist die MSC unabhängig von der Implementation der Modalitäten und kann somit flexibel gestaltet werden. Jede Modalität muss ihre eigene Servicebeschreibung zur Verfügung stellen, denn dadurch kann die MSC die Menge aller nutzbaren Modalitäten, pro EMODE Anwendung, ermitteln. Aus dieser Menge werden die am besten geeigneten Modalitäten, im aktuellen Kontext der jeweiligen EMODE Anwendung, bestimmt.

Folgende Informationen müssen in der Beschreibung einer Modalität enthalten sein:

1. Kennzeichnung als Dienstkomponente (Abgrenzung gegenüber Dienstnutzer)
2. Angaben über das Ein-/Ausgabegerät
3. Identifikation
4. Ein-/Ausgabemodus (GUI, Sprache)
5. Ein-/Ausgabemedium

Zusätzlich können noch weitere Elemente in der Beschreibung enthalten sein, z. B. Angaben über Eingabegrammatiken bzw. über besondere Ausgabefunktionen. Diese sind jedoch stark abhängig von der jeweiligen Anwendung und können nicht allgemein festgelegt werden.

2.3. OSGi – Framework für Service-orientierte Anwendungen

Die *Open Service Gateway Initiative* beschreibt eine Spezifikation für die Gestaltung, Entwicklung und Auslieferung von Dienstkomponenten, innerhalb einer Serviceplattform. Sie besteht aus 2 Teilen, dem *OSGi Framework* und einer Menge von Standarddiensten. Das *OSGi Framework* ist für die Auslieferung zuständig und dient den Dienstkomponenten als Ausführungsumgebung. Es ist ein Java *Framework* und bietet daher den Vorteil, dass es unabhängig von der jeweiligen Plattform eingesetzt werden kann. Es wird speziell für die Entwicklung Service-orientierter, verteilter Anwendungen genutzt.

Das *Framework* realisiert ein Komponentenmodell, einen Mechanismus zur Registrierung der Dienste und die Auslieferung der Dienstkomponenten. Der Einsatzbereich beschränkt sich nicht nur auf vernetzte Umgebungen, wie bspw. die sog. *Home-Area Networks* (vernetztes Haus) oder die Vernetzung von Geräten innerhalb eines Autos, sondern kann auch auf Anwendungsumgebungen erweitert werden. Dort ist es besonders geeignet zur Realisierung von *Multi-Plugin* Anwendungen. Ein Beispiel dafür ist die Integration des *OSGi Frameworks* „Equinox“ in die *Eclipse*-Entwicklungsumgebung. Dies zeigt wie flexibel und vielseitig verwendbar das *Framework* ist.

Die Funktionalität des *Framework*'s wird in folgende Schichten unterteilt (siehe Abb. 2.6):

- Ebene der Sicherheit
- Ebene der Module
- Ebene des Lebenszyklus
- Ebene der Services
- Ebene der Aktualisierung

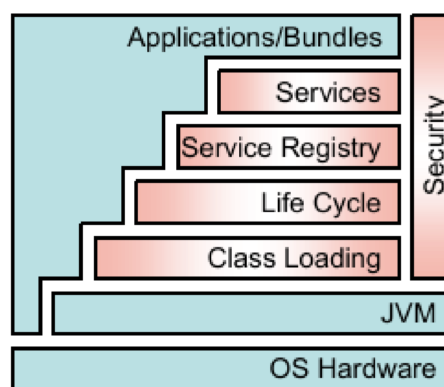


Abbildung 2.6.: OSGi Schichtenarchitektur [OSG05]

Die folgenden Abschnitte 2.3.1 und 2.3.2 sollen das *OSGi Framework* genauer charakterisieren.

2.3.1. Servicekomponenten – Bundles

Eine Komponente, die bestimmte Dienste anbietet, wird in der OSGi Spezifikation als *Bundle* (Bündel) bezeichnet. Ein Bundle enthält, neben Java *Class*-Dateien, zusätzliche Ressourcen, z. B. Konfigurationsdateien, HTML-Dateien und weitere. Die Ressourcen können beliebiger Art sein.

In der OSGi Serviceplattform können nur *Bundles* für die Ausführung und Auslieferung von Java-basierten Anwendungen genutzt werden. Ein Bundle wird als Java Archiv (JAR) zusammengestellt. Die JAR-Datei des *Bundle* muss, neben anderen, folgende 2 wichtige Informationen in einer Manifestdatei enthalten.

Bundle-Classpath Dieser *Manifest-Header* erlaubt einem *Bundle* einen intern eingebetteten Klassenpfad zu deklarieren. Mit diesem können Klassen auf integrierte Java Archive zugreifen, z. B. um weitere *Frameworks* im *Bundle* nutzen zu können. Dieser Klassenpfad ist bei mehreren Ressourcen als Liste, deren Elemente durch ein Komma voneinander getrennt werden, realisiert.

Bundle-Activator Diese *Header*-Information identifiziert die Java Klasse, die das *Interface BundleActivator* implementiert. Die Information ist entscheidend für den Lebenszyklus eines *Bundle*.

Weitere Informationen zur Implementation der *Bundle* werden in der Spezifikation zum *Framework* [OSG06] beschrieben.

2.3.2. Besondere Eigenschaften

Dynamische Natur

Im technischen *Whitepaper* [OSG05] wird folgendes ausgesagt:

„Ein Experte in der komponenten-orientierten Programmierung bezeichnet die *OSGi* Spezifikation als ‚Java's best kept secret' . Sein Enthusiasmus gegenüber der *OSGi* Serviceplattform leitet sich von der dynamischen Natur des *OSGi Framework* ab. "

Die dynamische Natur des *OSGi Framework* bezieht sich auf die Handhabung der Komponenten zur Laufzeit. Die Installation neuer *Bundle*, die Registrierung neuer Dienste oder die Aktualisierung existierender *Bundle* erfordert keinen Neustart der virtuellen Maschine (Java Virtual Machine, JVM).

Modularisierung

Durch die Kapselung der *Bundle*-Komponenten, gegenüber dem *Framework*, weist die Architektur Eigenschaften wie Flexibilität und Erweiterbarkeit auf. Die *Bundle* können durch Import/Export-Zugriffe gegenseitig Klassen austauschen. Dies erlaubt das Bereitstellen von Bibliotheken für andere *Bundle*-Komponenten. Diess bedeutet, dass die Ressourcen geteilt werden können und somit Speicherplatz eingespart wird (2.7).

Da das Framework komplett in Java implementiert ist, kann es auf unterschiedlichen Plattformen eingesetzt werden. Diese Eigenschaften machen das *OSGi Framework* interessant für die Entwicklung einer Service-orientierten Laufzeitumgebung nach der EMODE Spezifikation.

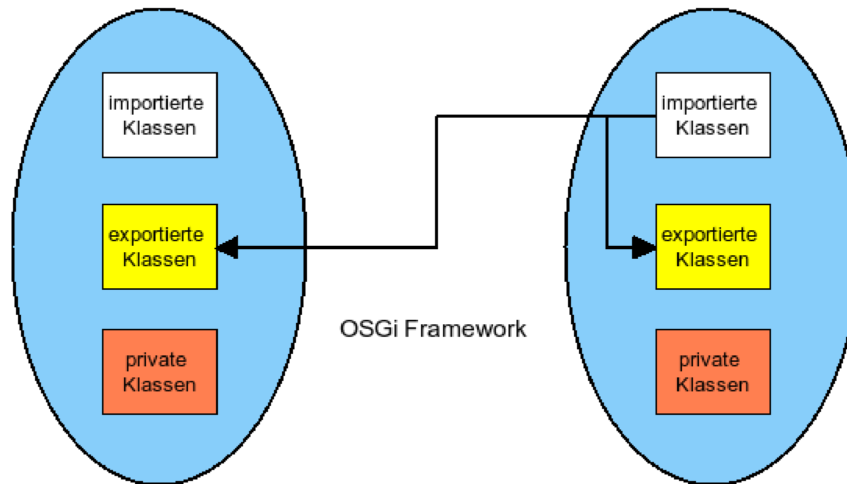


Abbildung 2.7.: Beziehungen der *Bundles* in OSGi

2.4. Die EMODE-Anwendungen

In diesem Abschnitt soll der Begriff der EMODE Anwendung näher erläutert und deren Eigenschaften, gegenüber denen einer herkömmlichen Anwendung, abgegrenzt werden.

Eine herkömmliche Applikation implementiert Benutzerschnittstellen direkt im eigenen Quellcode. Hauptsächlich handelt es sich dabei um sog. *GUI's*, die durch spezielle Funktionsbibliotheken oder sog. *Toolkits* bereitgestellt werden, z. B. die Win32-API, QT oder Java Swing. Der Entwickler der Anwendung setzt dabei generell das Vorhandensein des Displays voraus.

Wie das Szenario aus der Einführung 1 zeigt, muss der Entwickler einer EMODE Anwendung damit rechnen, dass sich die Verfügbarkeit der Ein- und Ausgabegeräte, wie z.B. das Display, ständig verändern kann. Es können neue Ein- und Ausgabegeräte zur Laufzeit hinzugefügt werden, die im Verlauf der EMODE Anwendung verwendet werden sollen. Die neuen Geräte und deren Funktionen sind der Anwendung unbekannt, d. h. die direkte Implementation von Ein- und Ausgabefunktionalitäten in der Anwendung ist nicht möglich. Die Entwicklung von EMODE Anwendungen verlangt ein neues Entwicklungskonzept - die multimodale Benutzerschnittstelle wird deklarativ beschrieben.

Eine EMODE Anwendung folgt dem *Modell-View-Controll* Paradigma. Die Applikationslogik, das Datenmodell und die Benutzerschnittstelle werden getrennt voneinander modelliert. Die Funktionen der Applikation greifen auf die Elemente des Datenmodells zu, verarbeiten diese und geben bestimmte Informationen durch die Benutzerschnittstelle nach außen. Die Eingabedaten werden durch die Benutzerschnittstelle an die Applikationslogik weitergegeben und durch diese in den Datenbestand integriert. Die EMODE Anwendungen geben selbst keine Implementationen von *User Interface* Elementen vor, sie beschreiben lediglich die Benutzerschnittstelle.

Die verschiedenen Eingabemodalitäten sind für die Aufnahme, die Verarbeitung und die Weiterleitung der Eingabedaten an die Anwendung zuständig. Sie geben die Daten in interpretierter Form an die EMODE Anwendungen weiter.

Jede EMODE Anwendung muss die relevanten Informationen aus dem Datenstrom der interpretierten Eingaben extrahieren können. Die Extraktion der Informationen erfolgt abhängig vom jeweiligen Kontext der Anwendung und muss durch sie einzeln durchgeführt werden.

Im folgenden Abschnitt 2.4.1 wird das Format für die Beschreibung der multimodalen Benutzerschnittstelle näher untersucht und diskutiert, welche Anforderungen es zu erfüllen hat. Zusätzlich soll die Struktur des abtrakten *User Interface* erläutert werden.

2.4.1. Beschreibung des *User Interface* von EMODE Anwendungen

In diesem Abschnitt wird die Beschreibung der multimodalen Benutzerschnittstelle diskutiert. Dabei werden zunächst die Anforderungen an die Beschreibung der multimodalen Benutzerschnittstelle erläutert und anschliessend ein existierendes Format vorgestellt. Es wird anhand der Anforderungen bewertet.

2.4.1.1. Anforderung an die Beschreibung des *User Interface*

Die Beschreibung des *User Interface* (UI) einer EMODE Anwendung muss folgende Anforderungen nach der EMODE Spezifikation [HHN] erfüllen.

1. Die darzustellenden Informationen müssen unabhängig, vom jeweiligen Ausgabegerät und der Ausgabemodalität, beschrieben werden.
2. Die abstrakte Beschreibung des *User Interface* muss in andere modalitätenspezifische Formate transformiert werden können.
3. Hinweise, die zur Berechnung der Zuordnung von darzustellenden Informationen zu den entsprechenden Ausgabemodalitäten genutzt werden, sollen in der Beschreibung des UI optional enthalten sein.
4. Hinweise, die zur Bestimmung alternativer Ausgabemodalitäten genutzt werden sollen, müssen optional in der Beschreibung des UI enthalten sein.
5. In der Beschreibung des UI sollen Grammatiken zur Interpretation spezieller Eingaben (z. B. Spracheingabe) enthalten sein.
6. Hinweise, die zur Bestimmung alternativer Eingabemodalitäten genutzt werden sollen, müssen optional in der Beschreibung des UI enthalten sein.
7. Bestimmte Elemente des UI müssen an die Funktionen der Anwendung gebunden werden können.
8. Bestimmte Elemente des UI sollen den Elementen des Datenmodells zugeordnet werden können.

2.4.1.2. D3ML

Die *Device Independent MultiModal Markup Language* (D3ML) dient der Modellierung multimodaler Benutzerschnittstellen von beliebigen Anwendungen. Das Ziel der Verwendung dieser Beschreibungssprache ist, die Meta-Informationen der Benutzerschnittstelle der Anwendung zu nutzen, um die UI-Dialoge an das entsprechende Ausgabegerät anzupassen. Dies bedeutet die Ausgaben sollen dynamisch generiert werden.

D3ML setzt sich grundsätzlich aus anderen Beschreibungssprachen zusammen und bietet zusätzliche Erweiterungen an. Hauptsächlich ist es als Erweiterung der Spezifikation nach XHTML 2.0 zu verstehen. Die Grundstruktur eines D3ML-Dokumentes entspricht der eines XHTML-Dokumentes. Die Erweiterungen lassen sich vorwiegend in den XHTML-Elementen `<head>` und `<body>` auffinden. Folgende Informationen werden im `<head>`-Element eines D3ML-Dokumentes eingebettet.

- Layout-Informationen
- Informationen der Ereignisbehandlung

- Informationen des Datenmodells

In Abbildung 2.8 wird dargestellt, welche Elemente im Bereich des `<head>`-Elementes integriert werden können.

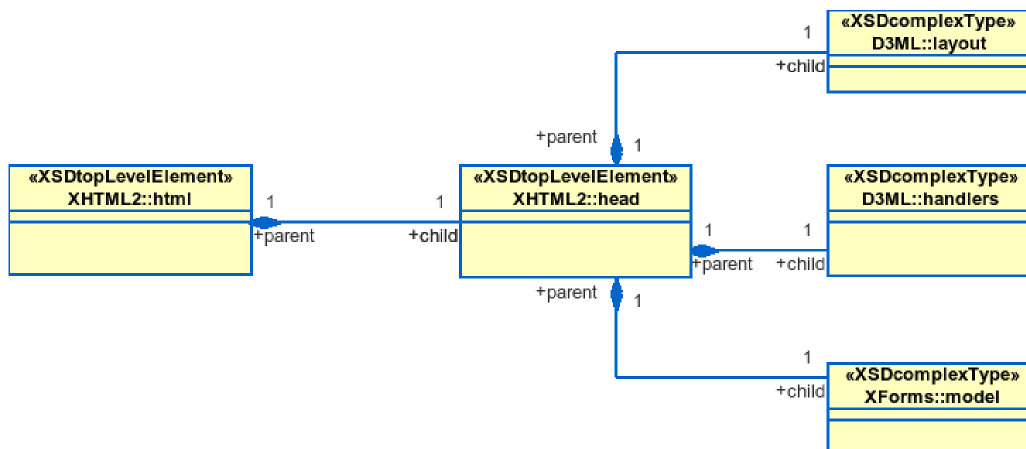


Abbildung 2.8.: D3ML `<head>`-Element [SAP06]

Folgende Informationen werden im `<body>`-Element eines D3ML-Dokumentes eingebettet.

- Informationen der Elemente des *User Interface*
- Informationen zu *Layout*-Sektionen

Die Abbildung 2.9 stellt die Elemente innerhalb des `<body>`-Elementes eines D3ML-Dokumentes dar. Alle *User Interface*-Elemente der XForms-Spezifikation [Boy07] werden in der Abbildung als `controls` bezeichnet.

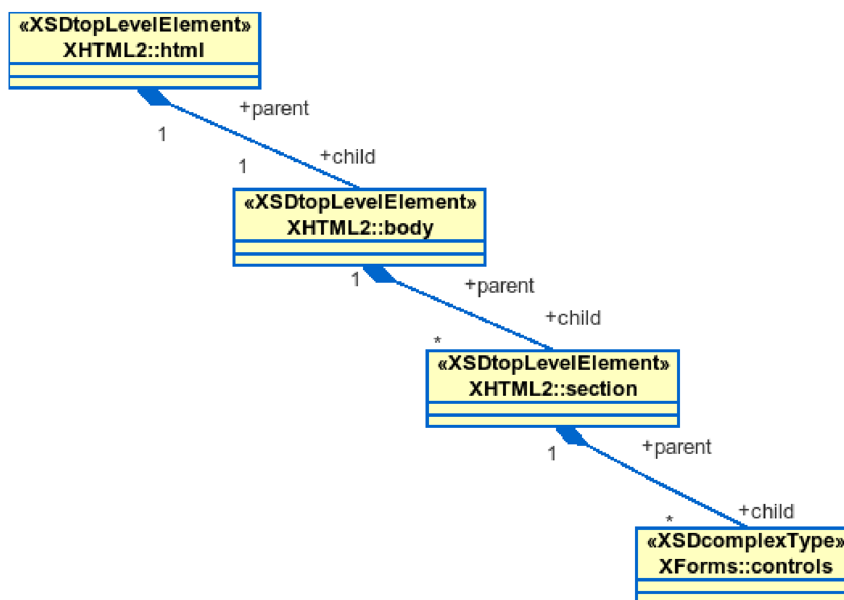


Abbildung 2.9.: D3ML `<body>`-Element [SAP06]

In den folgenden Punkten werden die Anforderungen aus dem Abschnitt 2.4.1.1 mit dem D3ML Format verglichen.

1. Das D3ML Format ist eine plattformunabhängige Beschreibung multimodaler Benutzerschnittstellen und erfüllt somit die 1. Anforderung.

2. Ein D3ML-Dokument enthält Informationen im XML-Format und kann daher durch Transformationen (XSLT) in modalitätspezifische Formate verändert werden.

3. Das D3ML Format bietet die Möglichkeit, die Bereiche der darzustellenden Informationen den einzelnen Ausgabemodalitäten zuzuordnen. Dies geschieht mit Hilfe der *DISelect* Spezifikation (siehe [SAP06] und [LMF07]). Innerhalb des `<d3ml:layout>` Elementes können die Informationsbereiche der Benutzerschnittstelle, durch das `<sel:select>` Element, so strukturiert werden, dass die verschiedenen Ausgabemodalitäten bestimmte Teile der Ausgabeinformationen ausgeben. Die Zuordnung der Bereiche der Ausgabeinformationen zu den Ausgabemodalitäten erfolgt über die Angabe bestimmter Attribute, die eine Modalität besitzen kann. Als Beispiel dient das Listing 2.1. Alle Modalitäten, die Informationen z. B. visuell darstellen, können die Elemente, die im entsprechenden `<sel:when>` Bereich enthalten sind, speziell ausgeben. In [SAP06] ist eine Übersicht der verwendbaren Attribute und der zugehörigen Auswahlkriterien enthalten.

4. Die 4. Anforderung kann ebenfalls durch die Beschreibungen der *DISelect* Spezifikation erfüllt werden. Im Listing 2.1 wird die Menge der Informationen der Sprachausgabemodalität alternativ zu der Menge der Informationen der visuellen Ausgabemodalität angegeben. Dabei muss beachtet werden, dass nicht die Informationen selbst, sondern die Bereiche, die die Ausgabeinformationen enthalten, relevant sind (die sog. *Layoutcontainer* des D3ML Format).

5. Das D3ML Format bietet die Möglichkeit Grammatikdateien zu referenzieren. Weiterhin ist es möglich Elemente anzugeben, die zu einer Grammatik hinzugefügt werden sollen.

6. Die *DISelect* Spezifikation der D3ML Dokumente bietet die Möglichkeit, die verschiedenen Eingabeelemente der multimodalen Benutzerschnittstelle alternativen Eingabemodalitäten zuzuordnen. Mit Hilfe des Auswahlkriteriums „inputSupport“ (siehe [SAP06]) können die Eingabeelemente der multimodalen Benutzerschnittstelle bestimmten Eingabemodalitäten zugeordnet werden. Weitere Erläuterungen dazu sind in [SAP06] enthalten.

7. Die Bindung der UI-Elemente an die Funktionen der Anwendung kann innerhalb des `<handler>` Elementes der D3ML Beschreibung erfolgen.

8. In D3ML können die Elemente der Benutzerschnittstelle und die Elemente des Datenmodells durch Referenzelemente, den `<bind>` Elementen und deren Attribute, oder durch das spezielle `ref`-Attribut eines UI-Elementes, verbunden werden.

Listing 2.1: Angabe alternativer Ausgabemodalitäten

```
1  <sel:select>
2    <sel:when expr="outputMethod() = 'visual'">
3      <d3ml:layout-container>
4          ...
5      </d3ml:layout-container>
6    </sel:when>
7    <sel:when expr="outputMethod() = 'voice'">
8      <d3ml:layout-container>
9          ...
10     </d3ml:layout-container>
11   </sel:when>
12 </sel:select>
```

Da die Integration des Datenmodells der Anwendung in das D3ML-Dokument besonders von Interesse ist, soll im nächsten Abschnitt die XForms-Spezifikation genauer betrachtet werden. Dabei liegt der Hauptaugenmerk auf der Beschreibung einer abstrakten Benutzerschnittstelle.

2.4.1.3. XForms - Die Bindung der UI-Elemente an die Applikationsdaten

XForms sind Nachfolger der HTML-Formulare. Sie sind jedoch so gestaltet, dass sie auch in anderen XML-Grammatiken genutzt werden können, wie bspw. D3ML. Die Abbildung 2.10 stellt die Integration der XForms-Elemente in D3ML schematisch dar. Die nachfolgenden Punkte charakterisieren das Datenmodell einer EMODE Anwendung, welches durch XForms-Elemente beschrieben wird. Die UI-Elemente der multimodalen Benutzerschnittstelle werden nach der XForms-Spezifikation [Boy07] als *Form Controls* bezeichnet und werden ebenfalls nachfolgend charakterisiert.

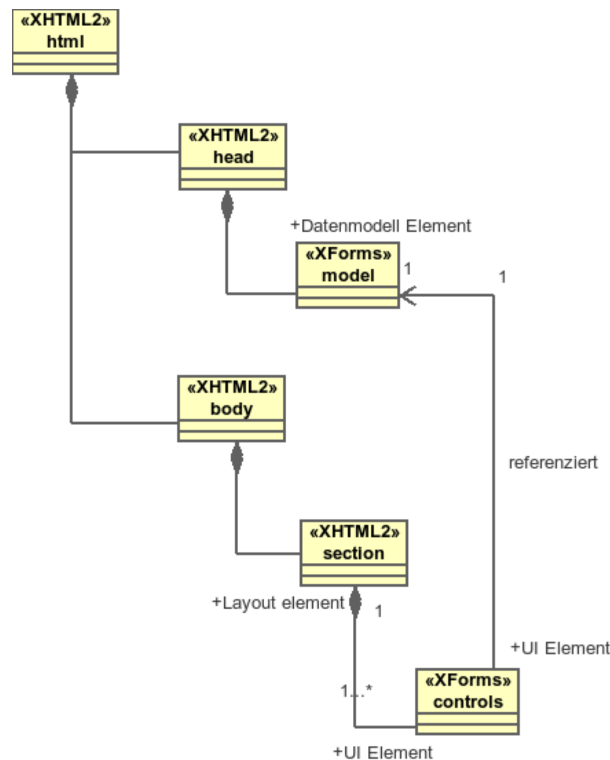


Abbildung 2.10.: Integration von XForms in D3ML

Das Datenmodell der EMODE Anwendung

Das Element `<model>` ist ein Container für die Elemente, die das Datenmodell einer Applikation enthalten. Das Datenmodell selbst kann dabei als Referenz auf eine externe Datei oder direkt eingebunden werden, es muss zumindest wohlgeformt im XML-Format vorhanden sein. Die Instanzdaten werden innerhalb des `<model>`-Elementes in `<instance>` abgelegt. Mittels XPath-Ausdrücken kann auf die Daten zugegriffen bzw. Verbindungen von UI-Elementen zu den zugehörigen Daten hergestellt werden. Als Beispiel dient ein Nachrichtenfenster (*Message Browser*) einer *Chat*-Applikation. Alle Nachrichten die im Message Browser dargestellt werden sollen, werden in einem dafür vorgesehenen Element des Datenmodells gespeichert. Dieses Element ist aufgrund einer eindeutigen Identifikation und mittels eines XPath Ausdrucks mit dem *Message Browser*-Element verbunden. Die genauen Details zum Datenmodell und der Verbindungen zu den UI-Elementen sind in [DSSR00] und [Boy07] beschrieben.

Die UI-Elemente der EMODE Anwendung – Form Controls

Elemente der Benutzerschnittstelle werden in XForms abstrakt beschrieben. Sie kapseln *high-level* Semantiken ohne die tatsächliche Implementation des jeweiligen UI-Elementes vorgeben zu müssen. Zum Beispiel stellt das Element `<input>` eine Möglichkeit der Eingabe von Daten dar, ohne eine spezielle Vorgabe der Darstellung und des zu verwendenden Darstellungsmediums. Funktionale Aspekte der *User-Interface*-Elemente werden von den Aspekten der Darstellung und des Verhaltens getrennt. Dadurch ist es möglich Anwendungen so zu gestalten, dass die Kommunikationsdialoge durch unterschiedliche Ausgabegeräte präsentiert werden können.

3. Entwurf

In diesem Kapitel wird der softwaretechnische Entwurf, basierend auf den Konzepten und Technologien des vorherigen Kapitels erläutert.

Zunächst wird der Gesamtentwurf der MSC vorgestellt, um einen Überblick über die einzelnen Elemente und deren Aufgaben zu ermöglichen. Im Anschluss werden die Entwurfsentscheidungen der Modalitäten erläutert. Die internen Strukturen werden genauer untersucht und diskutiert. Der darauf folgende Abschnitt enthält den Entwurf der EMODE Anwendungen. Dabei wird das Konzept des Aufrufs der Anwendungsfunktionen entwickelt. Die Prozesse der Eingabeinterpretation und Ausgabeadaption werden im nächsten Schritt diskutiert. Das Verhalten der Komponenten zur Laufzeit wird anschliessend modelliert. In Anbetracht der Aufgabenstellung werden spezielle Modalitäten,wie z.B. die Modalität der Spracherkennung, mit einbezogen.

3.1. Der Gesamtentwurf der MSC

Die MSC hat die Aufgabe eine Anwendung mit den verschiedenen Ein- und Ausgabegeräten zu verknüpfen und wenn nötig die Verknüpfung auch wieder aufzulösen. Des Weiteren soll sie mehrere Anwendungen und Modalitäten parallel verwalten können. Die folgende Abbildung 3.1 soll die Beziehung der MSC zu den EMODE Anwendungen und den Modalitäten verdeutlichen.

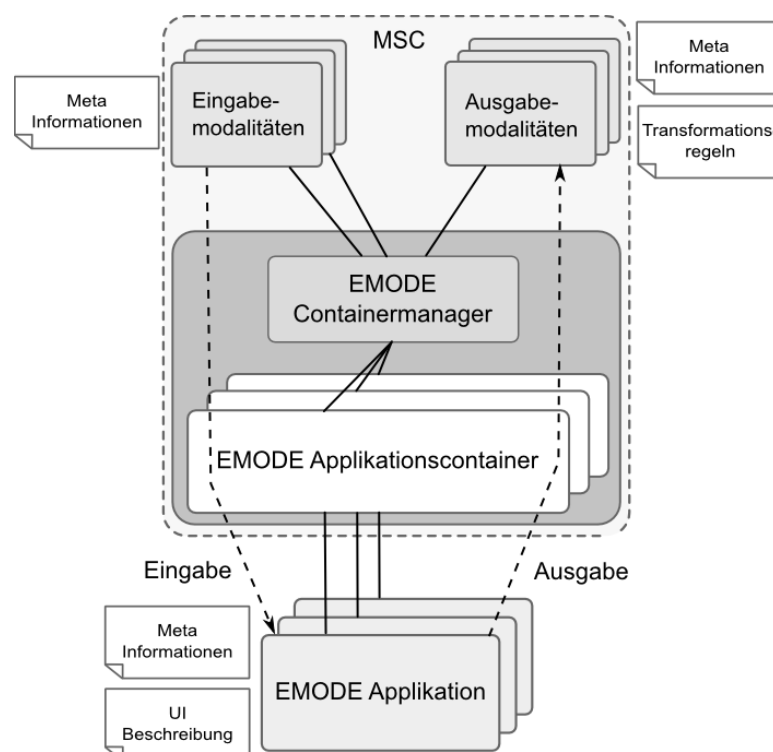


Abbildung 3.1.: Überblick Modalitäten-MSC-EMODE Applikation

Die Modalitäten besitzen spezielle Beschreibungen, die an die MSC übergeben werden. Die Meta-Informationen der Modalitäten werden im noch folgenden Abschnitt 3.2.2 genauer erläutert.

Jede Ausgabemodalität muss nach Abschnitt 2.1.4 eine Anpassung der Ausgabeinformationen durchführen. Zur Adaption der Informationen, an das entsprechende Ausgabegerät, muss die Modalität die Informationen transformieren, dazu benötigt sie Transformationsregeln, die die Umsetzung der Ausgabenachrichten in das jeweilige Ausgabeformat beschreiben. In dieser Arbeit sollen die Transformationsregeln in sog. *XSLT-Stylesheets* Dateien enthalten sein. Die Prozesse innerhalb der Ausgabemodalitäten werden in 3.2.1 noch genauer diskutiert.

Da die MSC mehrere Anwendungen verwalten soll und da jede Anwendung unabhängig von der MSC existiert, soll eine Anwendung innerhalb der MSC durch einen Container repräsentiert werden. Der Anwendungscontainer, auch als EMODE Applikationscontainer bezeichnet, kann als Anwendungsdomäne innerhalb der MSC angesehen werden. Die Container sind dafür zuständig, die Daten der Anwendungen, z.B. Meta-Informationen oder weitere zusätzliche Dateien innerhalb der MSC zu halten und für weitere Verarbeitungsschritte bereitzustellen.

Damit die MSC neue Anwendungen registrieren und für diese neue Container bereit stellen kann, benötigt sie eine zentrale Komponente, den EMODE Containermanager. Er ist hauptsächlich für die Koordination der Container und der zugehörigen EMODE Anwendungen zuständig. Er muss zusätzlich die Ein- und Ausgabemodalitäten mit den Anwendungscontainern verknüpfen. Nachdem die Modalitäten und die Anwendungen verknüpft worden, ist der Anwendungscontainer für die Koordination der Kommunikation zuständig.

3.1.1. Der Entwurf der MSC im Detail

Die folgenden Elemente dienen der Beschreibung wichtiger Klassen, die in der Abbildung A.1 in UML-Notation dargestellt werden.

EMODEContainerManager

Dieses Objekt verwaltet die `EMODEApplicationContainer`-Objekte, die für die Kommunikation mit den EMODE Anwendungen zuständig sind. Es kann durch Helferobjekte feststellen wann eine EMODE Anwendung gestartet wird und die Ereignisse an den dafür zuständigen `EMODEApplicationListener` delegieren. Wird eine Anwendungen beendet, so wird das Ereignis ebenfalls übergeben.

Durch das Objekt des `EMODEServiceListener` können aktivierte Modalitäten registriert werden. Der `EMODEContainerManager` besitzt den Zugriff auf alle aktivierten Modalitäten und auf alle Applikationscontainer. Er kann eine Verbindung von der Modalität zur laufenden Anwendung und umgekehrt herstellen.

Nach dem Konzept der SOA ist dieses Objekt dafür zuständig, die angeforderten Modalitäten (Services) zu lokalisieren und dem *Service Requester* (aktuelle Anwendung) bereitzustellen. Es besitzt somit die Funktionen eines *Service Locator*.

EMODEApplicationContainer

Das Objekt ist für die Kommunikation mit einer zugehörigen EMODE Anwendung verantwortlich. Die Anwendung selbst läuft in einem eigenständigen Prozess und nutzt Mittel der Interprozesskommunikation (IPC), um die Ein- und Ausgabedaten mit dem Container-Objekt auszutauschen.

Der `EMODEApplicationContainer` stellt dem `EMODEContainerManager` die Informationen der Anwendung zur Verfügung, z. B. die D3ML-Beschreibung der Benutzerschnittstelle oder Informationen über die Eigenschaften der zu nutzenden Modalitäten. Wird dem `EMODEApplication-Container` eine passende Modalität zugeordnet, so kann sie durch ihn innerhalb des `Modality-Connection`-Objektes genutzt werden, d. h. alle interpretierten Eingabedaten des Nutzers werden durch ihn an die Anwendung übergeben. Alle Ausgabedaten der Anwendung werden an die entsprechende Ausgabemodalität, über den Container, weitergeleitet. Die genauen Abläufe der Interpretation der Eingabe und der Adaption der Ausgabe wird jeweils in einem eigenen Abschnitt in diesem Kapitel noch diskutiert und soll hier nicht weiter vertieft werden (siehe 3.4 und 3.5).

Der `EMODEApplicationContainer` spielt für die Anwendung innerhalb der MSC eine wichtige Rolle, besonders während der Adaption der Ausgabenachrichten ist er für die Sicherung der Ein- und Ausgabedaten innerhalb des Datenmodells der Anwendung zuständig. Im Abschnitt 3.1.2 werden die internen Prozesse und die Struktur näher erläutert.

EMODEAppContainerFactory

Wird eine Anwendung vom `EMODEContainerManager` erkannt, so muss durch dieses Objekt ein neuer `EMODEApplicationContainer` erzeugt werden. Das Erstellen des Container-Objektes einer EMODE Anwendung wird innerhalb der Erzeugermethode `createEMODEAppContainer` gekapselt. Dadurch können mögliche Erweiterungen leichter vorgenommen werden.

EMODEEndPoint

Die Kommunikation des `EMODEApplicationContainer` mit einer EMODE Anwendung erfolgt über dieses Objekt. Mit Hilfe des `EMODEEndPoint` kann das Container-Objekt Lese- und Schreibeoperationen durchführen. Dazu muss das Objekt in einem eigenen Kommunikationsteilprozess (`Thread`) gestartet werden.

Der `EMODEEndpoint` stellt eine Schnittstelle für die Implementierung der IPC zur Verfügung. Die IPC wird in dieser Arbeit durch `Sockets` realisiert. Dadurch kann die Grenze der verschiedenen Programmiersprachen überwunden werden und bietet somit die Möglichkeit der offenen und flexiblen Gestaltung der MSC.

EMODEEndPointFactory

Die Erzeugung der `EMODEEndpoint`-Objekte erfolgt durch die Klasse der `EMODEEndPointFactory`. Dabei werden die nötigen IPC Objekte vom `EMODEContainerManager` an die Fabrikklasse übergeben. Die `EMODEEndPointFactory` wird während des Prozess der Registrierung einer startenden EMODE Anwendung eingesetzt. Die erzeugten `EMODEEndPoint`-Objekte werden an den zugehörigen `EMODEApplicationContainer` übergeben.

EMODEApplicationListener

Während des Startprozesses baut eine EMODE Anwendung eine Verbindung zum `EMODEContainer-Manager` auf. Dabei wird ein Ereignis ausgelöst und an das Objekt des `EMODEApplication-Listener` übergeben. Daraufhin werden durch das Objekt alle nötigen Schritte für die Registrierung der EMODE Anwendung im System vorgenommen. Dazu gehört die Erstellung des

`EMODEApplicationContainer`-Objektes und die Bereitstellung der Kommunikationsmittel für die dazugehörige EMODE Anwendung.

EMODEModalityApplicationConnector

Das Objekt übernimmt die Aufgabe, die registrierten EMODE Anwendungen mit bereitstehenden Modalitäten zu verknüpfen. Der Prozess des Verknüpfens setzt sich aus folgenden 2 Teilschritten zusammen.

1. Vergleich der Ein- und Ausgabeeigenschaften der aktuellen Modalität mit den angeforderten Ein- und Ausgabeeigenschaften der EMODE Anwendung
2. Verknüpfung der aktuellen Modalität mit der jeweiligen EMODE Anwendung nach erfolgreichem Vergleich

Eine weitere Aufgabe des Objektes der Klasse `EMODEModalityApplicationConnector` ist das Lösen der Verknüpfung zwischen der EMODE Anwendungen und der aktuellen Modalitäten. Dies geschieht wenn eine Modalität deaktiviert bzw. wenn eine EMODE Anwendung beendet wird.

EMODEServiceListener

Startende Modalitäten werden durch das Objekt der Klasse `EMODEServiceListener` erkannt und an den `EMODEContainerManager` übergeben. Die Klasse implementiert die OSGi Schnittstelle `ServiceListener` und kapselt den Mechanismus der Registrierung und Deregistrierung von Modalitäten-*Bundle* im *OSGi-Framework*.

3.1.2. Der Anwendungscontainer

Eine EMODE Anwendung wird durch den Anwendungscontainer innerhalb der MSC repräsentiert. Er ist für die Koordination der Kommunikation der Anwendung mit den zugeordneten Modalitäten zuständig. Eine weitere Aufgabe ist die Bereitstellung von notwendigen Anwendungsdaten, z. B. die Anforderungsbeschreibung, um passende Modalitäten zu ermitteln und um diese der Anwendung zuordnen zu können.

Das folgende Aktivitätsdiagramm 3.2 soll die grundlegenden Prozesse innerhalb des Anwendungscontainers darstellen.

Der Entwurf des Anwendungscontainers ist bedingt durch die Tatsache, dass auch Anwendungen unterstützt werden sollen, die nicht in der Programmiersprache der MSC implementiert wurden. Der Aufruf der Anwendungsfunktionen kann daher nicht direkt durch den Applikationscontainer erfolgen. Das Problem soll im Abschnitt der EMODE Anwendungen 3.3 näher untersucht werden.

Der Applikationscontainer empfängt die Eingabedaten der verschiedenen Eingabemodalitäten. Die Vereinigung der Eingabeinformationen nach 2.1.3 kann innerhalb des Containers durchgeführt werden. In dieser Arbeit soll die Vereinigung der interpretierten Eingabedaten nicht weiter beachtet werden, daher ist der Anwendungscontainer nur für die Untersuchung der Eingabedaten nach bestimmten Datenelementen und für die Weiterleitung der Eingabeinformationen an die zugehörige Anwendung zuständig.

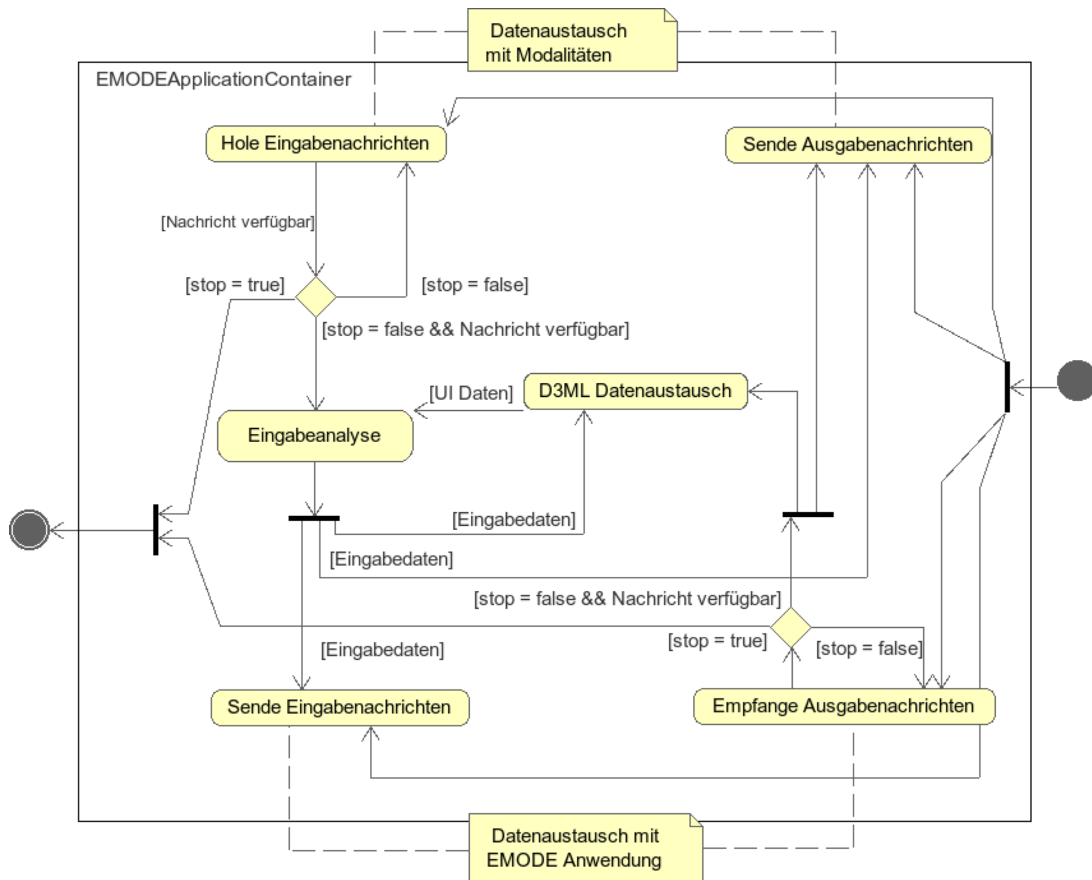


Abbildung 3.2.: Prozesse innerhalb des Applikationscontainer

Eine weitere Aufgabe des Applikationscontainers ist die Sicherung des Interaktionszustandes, in dem sich die Anwendung momentan befindet. Er muss die Änderungen der Benutzerschnittstelle, die durch die Benutzereingaben und Anwendungsausgaben auftreten, an die Beschreibung der multimodalen Benutzerschnittstelle weitergeben. Durch die Aktualisierung der multimodalen Benutzerschnittstelle können neu hinzukommende Modalitäten in den momentanen Interaktionszustand der Anwendung versetzt werden.

Welche Elemente der Benutzerschnittstelle verändert werden sollen, wird durch die Anwendung oder durch spezielle Eingabedaten der Eingabemodalitäten bestimmt. Im folgenden Abschnitt 3.1.2.2 soll geklärt werden, welche speziellen Datenelemente verwendet werden können.

Jede Anwendung besitzt eine Beschreibung der multimodalen Benutzerschnittstelle im D3ML-Format. Diese sollen im Verlauf der Anwendung durch den Anwendungscontainer modifiziert werden. Der Anwendungscontainer erhält während des Startprozesses der Anwendung eine Referenz auf die D3ML-Dokumente. Mit Hilfe der Referenz kann der Container auf die verschiedenen D3ML-Dateien zugreifen.

Die Analyse der Eingabedaten (siehe Abb. 3.2) dient der Ermittlung und Ausführung bestimmter Anweisungen, die im Datenstrom der Benutzereingabe enthalten sein können. Dabei kann es sich um das Einfügen von Eingabeinformationen in die multimodale Benutzerschnittstelle handeln oder um das Abfragen von Informationen, die bereits in der multimodalen Benutzerschnittstelle enthalten sind. Nach dem die Anweisungen durch die MSC ermittelt wurden, werden sie an eine interne Verarbeitungskomponente übergeben. Diese führt die entsprechenden Operationen innerhalb der D3ML-Dokumente durch. Das Objekt, welches für die Analyse der Eingabedaten verantwortlich ist, wird in der Abbildung A.2 als `EMODEInputMessageProcessor` bezeichnet.

Erzeugt die Anwendung Ausgabeinformationen, so müssen diese in die multimodale Benutzerschnittstelle integriert werden. Sie gelangen durch die Interprozesskommunikation zum Anwendungscontainer und werden hier ebenfalls in Form von speziellen Anweisungen an eine Verarbeitungskomponente übergeben. Diese führt die Modifikationen an der aktuellen Benutzerschnittstelle durch.

Im Klassendiagramm, in A.2, wird das Objekt durch die Klasse des `D3MLDBManager` repräsentiert. Es hat die Aufgabe, die Modifikationsanweisungen der Eingabemodalitäten, bzw. der EMODE Anwendungen, zu verarbeiten und die Informationen in das Datenmodell der Anwendung, welches in den D3ML-Dokumenten enthalten ist, zu integrieren.

In den Abschnitten 3.1.2.1 und 3.1.2.2 wird die Modifikation der multimodalen Benutzerschnittstelle näher erläutert. Zusätzlich wird das Format der Modifikationsanweisungen in 3.1.2.2 vorgestellt.

3.1.2.1. Die Modifikation der mulitmodalen Benutzerschnittstelle durch die Eingaben des Benutzers

Im Verlauf einer herkömmlichen Anwendung bestehen die Eingabedaten des Nutzers nicht nur aus Informationen die von der Benutzerschnittstelle an die Funktionen der Anwendung übergeben werden, sondern der Nutzer manipuliert die Benutzerschnittstelle und kombiniert dies mit eingegebenen Informationen. Dies ist der Fall wenn z. B. eine *Checkbox* angeklickt und zusätzlich etwas in ein Textfeld geschrieben wird. Eine herkömmliche Benutzerschnittstelle sichert die Manipulationen intern, d. h. im Fall der *Checkbox* wird der aktuelle Zustand durch eine interne Variable repräsentiert.

Eine herkömmliche Anwendung kann die Zustände der einzelnen UI-Elemente bei Bedarf abfragen und in Kombination mit den Eingabeinformationen verschiedene Funktionen ausführen. Nun stellt sich folgende Frage. Wie verhält sich eine multimodale Benutzerschnittstelle, die die UI-Elemente ausschließlich deklarativ beschreibt und selbst keine Funktionen zur Sicherung interner Zustände besitzt? Die Antwort auf diese Frage ist folgende. Der Entwickler der multimodalen Benutzerschnittstelle muss die Vorgänge zur Sicherung der internen Zuständen ebenfalls beschreiben. Dazu werden spezielle *Markup*-Elemente in der Beschreibung der multimodalen Benutzerschnittstelle eingesetzt, die im Abschnitt 3.4.2 vorgestellt werden.

Im Fall der *Checkbox* wird neben der Struktur auch die Aktion beschrieben, die einen Zustandswechsel auslöst. Der gegenwärtige Zustand der *Checkbox* wird in der multimodalen Benutzerschnittstelle mit Hilfe eines zugeordneten Datenelementes gesichert. (siehe Listing 3.1)

Listing 3.1: deklarative *Checkbox*-Beschreibung

```
<xforms:model>
   <xforms:instance id="checkbox">
      <flag>0</flag>
   </xforms:instance>
</xforms:model>
...
<xforms:trigger id="checker">
  <xforms:label>my checkbox</xforms:label>
  <xforms:action ev:event="DOMActivate">
    <xforms:setvalue
     ref="instance('ckeckbox')"
     nodeset="/flag"
     value="1"/>
  </xforms:action>
</xforms:trigger>
```

Im Fall einer multimodalen Anwendung ist die Sicherung der Zustände der einzelnen UI-Elemente notwendig, damit die verschiedenen Modalitäten in einem konsistenten Interaktionszustand verbleiben. Die multimodale Benutzerschnittstelle dient dabei als Ausgangspunkt für die Erhaltung des Interaktionszustands der einzelnen Modalitäten.

Im Fall der EMODE Anwendungen, wird die Beschreibung der multimodalen Benutzerschnittstelle als D3ML-Dokument an den Anwendungscontainer übergeben. Er stellt die multimodale Benutzerschnittstelle jeder, der Anwendung zugeordneten, Modalität zur Verfügung. Jede Modalität erzeugt aus der Beschreibung der multimodalen Benutzerschnittstelle ein geeignetes *User Interface* im Kommunikationskanal. Der Begriff *User Interface* soll nachfolgend, für die konkrete Darstellung der multimodalen Benutzerschnittstelle durch eine Modalität, verwendet werden.

Eine visuelle Ein-/Ausgabemodalität erzeugt aus der multimodalen Benutzerschnittstelle, z.B., ein entsprechendes HTML *User Interface* und präsentiert die Informationen mit Hilfe eines Browsers. Der genaue Ablauf der Anpassung der multimodalen Benutzerschnittstelle an die Darstellungsform der Ausgabemodalität wird in Abschnitt 3.5 erläutert.

Nachdem die Modalitäten die multimodale Benutzerschnittstelle in ein eigenes *User Interface* überführt haben, können sie zur Ausgabe von Informationen oder zur Aufnahme von Eingaben durch die Anwendung genutzt werden.

Im Fall der Eingabemodalitäten müssen alle Eingabedaten und Veränderungen des *User Interface*'s, der jeweiligen Modalität, an die multimodale Benutzerschnittstelle übergeben werden. Die multimodale Benutzerschnittstelle der EMODE Anwendung wird durch den Anwendungscontainer verwaltet. Die Eingabemodalitäten leiten die Veränderungen des *User Interface* an den Anwendungscontainer weiter.

Die Manipulationen der Benutzerschnittstelle werden, wie vorher geschildert (siehe Listing 3.1), durch Aktionen beschrieben. Die Eingabemodalitäten besitzen durch die Übernahme der multimodalen Benutzerschnittstelle Kenntnisse über die entsprechenden Aktionen der einzelnen UI-Elemente. Die Eingabemodalität muss die Aktionsbeschreibungen auswerten und gegebenfalls mit Zustandswerten der UI-Elemente versehen. Die Zustandswerte können unterschiedlicher Natur sein, sie zeigen an, ob z.B. eine *Checkbox* angeklickt wurde oder sie enthalten Eingabedaten des Benutzers. Innerhalb eines HTML *User Interface*'s kann der Zustandswert eines UI-Elementes über die eindeutige Identifikation ermittelt und in die Aktionsbeschreibung integriert werden. Als Beispiel dienen die Eingaben eines Textfeldes, die in die multimodale Benutzerschnittstelle eingefügt werden sollen.

Die Eingabeinformationen können nun durch die Modalität interpretiert und an den Anwendungscontainer gesendet werden. Er führt anhand der Beschreibung der Aktionen Operationen im D3ML-Dokument der multimodalen Benutzerschnittstelle durch und sichert somit den Zustand des veränderten UI-Elementes. Die Änderungen der multimodalen Benutzerschnittstelle werden nun wiederum durch den Anwendungscontainer an die restlichen Modalitäten weitergeleitet. In 3.4.2 werden die einzelnen Schritte der Eingabeverarbeitung näher erläutert.

3.1.2.2. Die Modifikation der multimodalen Benutzerschnittstelle durch die EMODE Anwendung

Eine EMODE Anwendung sendet spezielle Anweisungen an die MSC, um die jeweiligen Ausgabeinformationen der Modalitäten zu verändern. Der Anwendungscontainer empfängt die Nachrichten der zugehörigen EMODE Anwendung und muss diese verarbeiten (siehe Abbildung 3.2).

Jeder EMODE Anwendung können gleichzeitig mehrere Ausgabemodalitäten zugeordnet sein. Jede einzelne Ausgabemodalität verarbeitet die Informationen der Anwendung unterschiedlich.

Die Informationen der Ausgabe werden durch spezielle Formate beschrieben, damit das jeweilige Ausgabegerät sie korrekt darstellen kann. Damit die Informationen durch ein Display via Browser visuell präsentiert werden können, müssen sie z. B. in HTML beschrieben werden.

Die Anwendung hat selbst keine Kenntnisse über die erforderlichen Ausgabeformate der jeweiligen Ausgabemodalitäten. Dieses Problem soll durch die Adaption der Ausgabeinformationen durch die MSC gelöst werden. (siehe 3.5). Eine Grundlage der Adaption ist die Beschreibung der Anwendungsausgaben durch ein allgemeines Format.

Die multimodale Benutzerschnittstelle ist im Grunde eine XML-Beschreibung der Struktur und der Anordnung der UI-Elemente, gekoppelt mit der XML-Beschreibung des Datenmodells der Anwendung. Zur Veränderung von XML-Dokumenten durch allgemeine Anweisungen existiert die *XUpdate*-Technologie. Darin ist das Format zur Beschreibung von möglichen Modifikationen eines XML-Dokumentes enthalten. Genaue Informationen zur *XUpdate*-Technologie können in [LM00] nachgeschlagen werden.

Die Ausgabenachrichten der Anwendung, im *XUpdate*-Format, dienen dem Hinzufügen, dem Entfernen und dem Verändern von Informationen. Die *XUpdate*-Modifikationsoperationen werden deklarativ in XML beschrieben. Die eigentlichen Operationen werden durch eine Komponente innerhalb der MSC, anhand der Modifikationsbeschreibungen, durchgeführt.

Das folgende Listing 3.2 zeigt das Hinzufügen von Datenelementen in die Beschreibung der multimodalen Benutzerschnittstelle.

Listing 3.2: Einfügen eines Client

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xu:modifications version="1.0"
3   xmlns:xlink="http://www.w3.org/1999/xlink"
4   xmlns:xu="http://www.xmldb.org/xupdate"
5   xlink:href="chatUI.d3ml">
6     <xu:append select="//*[@id='m_clientlist']//clientlist">
7        <client>
8          <client-name>Carsten</client-name>
9          <client-id>2</client-id>
10       </client>
11    </xu:append>
12 </xu:modifications>
```

Das betreffende D3ML-Dokument wird durch das Attribut `xlink:href` im `<xu:modifications>` Element angezeigt. Alle Kindelemente von `<xu:modifications>` beschreiben verschiedene Operationen, die im jeweiligen D3ML-Dokument durchgeführt werden sollen.

Die Selektion von vorhandenen Elementen wird durch einen *XPath*-Ausdruck angegeben. Im Beispiel wird ein neues Datenelement in ein bereits vorhandenes Element eingebettet. Das `select`-Attribut der Modifikationsbeschreibung bestimmt vorhandene Datenelemente. Weitere Elemente der *XUpdate*-Modifikationsbeschreibung werden in [LM00] näher erläutert.

Mit Hilfe der *XUpdate*-Modifikationen kann eine EMODE Anwendung die Veränderungen innerhalb der Benutzerschnittstelle allgemein beschreiben und durch die MSC ausführen lassen. Die MSC kann die Modifikationsoperationen ohne Rücksicht auf den Zustand der Anwendung ausführen, d. h. der eigentliche Zweck der Modifikation der Benutzerschnittstelle bleibt der MSC verborgen. Die Anwendung bestimmt die Modifikationen, die MSC führt sie lediglich aus. Somit ist die Unabhängigkeit der MSC gegenüber der EMODE Anwendung gewährleistet und daher können mehrere EMODE Anwendungen parallel verwaltet werden.

Im Beispiel 3.2 werden neue Datenelemente an die Benutzerschnittstelle übergeben. Dabei ergibt sich folgendes Problem. Die Daten, die in die multimodale Benutzerschnittstelle integriert

werden sollen, sind stark von der Anwendungssituation abhängig. Im Beispiel wird ein *Client*-Element in ein Element des Datenmodells eingefügt. Durch die Bindung des Datenmodellelementes an ein Element der Benutzerschnittstelle wird diese ebenfalls implizit aktualisiert. Für die Beschreibung der multimodalen Benutzerschnittstelle im D3ML-Format ist es nicht unbedingt relevant wie ein zu integrierendes Datenelement gestaltet ist, es muss hauptsächlich wohl geformt im XML-Format vorliegen. Eine Ausgabemodalität kann die Bindung, zwischen Datenelement und dem Element der Benutzerschnittstelle, nur durch die Transformation in das Ausgabeformat übernehmen. Soll das *Client*-Element des vorherigen Beispiels 3.2 in eine visuelle Repräsentation, im HTML-Format, überführt werden, so muss die Ausgabemodalität das Element transformieren. Da neue Datenelemente anwendungsabbhängig sind und daher nicht durch die modalitätenspezifischen Transformationsregeln erfasst werden können, muss die EMODE Anwendung anwendungsspezifische Transformationen an die Ausgabemodalitäten verteilen. Damit ergibt sich folgende Frage. Wie kann die EMODE Anwendung anwendungsspezifische Transformationen an die jeweiligen Ausgabemodalitäten übergeben, wenn die Anwendung keine direkten Vorkenntnisse über die zu verwendenden Ausgabemodalitäten besitzt? Im Abschnitt 3.3.3.1 wird die Lösung des Problems erläutert.

3.2. Die Modalitäten

Nach dem Konzept des multimodalen Systems 2.1.1 bilden die Modalitäten die Repräsentation der Ein- und Ausgabegeräte auf der Ebene der Software. Im Aspekt der SOA können sie als Anbieter von Eingabe- bzw. Ausgabediensten angesehen werden.

Dieser Abschnitt soll sich den Modalitäten und ihrem Entwurf widmen. Zunächst wird die Funktionsweise der Modalitäten, je nach Kategorie, vorgestellt. Darauf folgend werden die Modalitäten im Aspekt der SOA diskutiert. Danach folgt der grundlegende Aufbau der Modalitäten, dabei werden weitere Teilkomponenten vorgestellt und näher erläutert. Zum Schluss dieses Abschnitts werden spezielle Eingabe- und Ausgabemodalitäten präsentiert.

3.2.1. Die grundlegenden Funktionsweisen der Modalitäten

Die verschiedenen Modalitäten können durch die folgende Kategorien näher charakterisiert werden.

Die Eingabemodalitäten

Die Eingabemodalitäten besitzen ausschließlich die Fähigkeit, die Eingaben des Benutzers, innerhalb eines Kommunikationskanals, verarbeiten zu können. Der Prozess der Verarbeitung der Benutzereingabe wird im folgenden Aktivitätsdiagramm 3.3 schematisch dargestellt.

Die Eingaben des Benutzers müssen nach Abschnitt 2.1.3 interpretiert werden, damit sie, unabhängig von welchem Eingabegerät sie stammen, durch die MSC verarbeitet werden können. Die interpretierten Daten werden im nächsten Schritt für die MSC bereitgestellt.

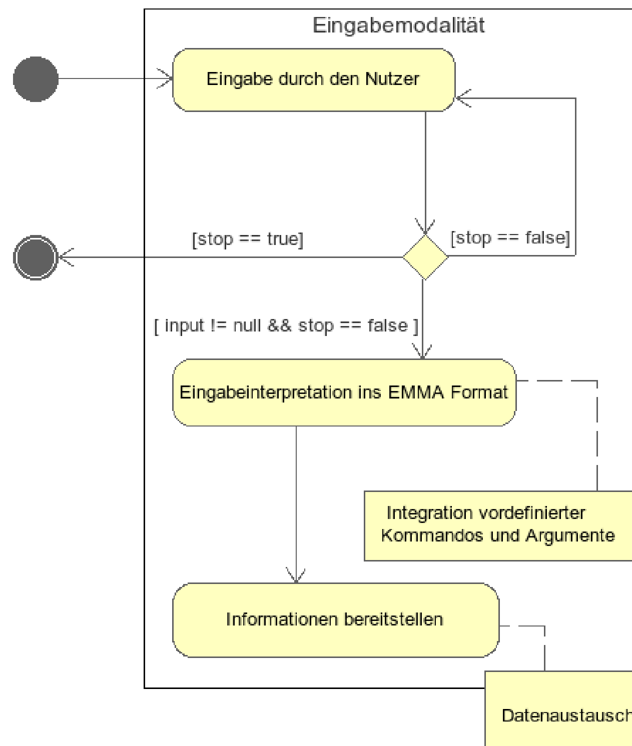


Abbildung 3.3.: Aktiviäten innerhalb der Eingabemodalität

Die Ausgabemodalitäten

Die Ausgabemodalitäten besitzen ausschließlich die Fähigkeit, die Ausgaben einer Anwendung, innerhalb eines Kommunikationskanals, verarbeiten zu können. Die Ausgaben einer Anwendung müssen nach Abschnitt 2.1.4 an das jeweilige Ausgabegerät angepasst werden.

Die verschiedenen Dialoge der Benutzerschnittstelle werden während des Aktivierens der Ausgabemodalität komplett in das Ausgabeformat der Modalität transformiert. Im weiteren Anwendungsverlauf werden neue Ausgabeinformationen jeweils einzeln transformiert und durch Modifikationsoperationen in den bestehenden Ausgabedialog der Modalität hinzugefügt. Das folgende Aktivitätsdiagramm 3.4 soll die Verarbeitung der Ausgabedaten veranschaulichen.

Da die Ausgabeinformationen abhängig von der Anwendung sind, kann die Ausgabemodalität diese nicht allein an das jeweilige Ausgabeformat anpassen. Die Anwendung muss daher für jedes mögliche Ausgabeelement eine spezielle Transformation anbieten und an die zu verwendende Ausgabemodalität übergeben. Im Abschnitt 3.3.3.1 wird der Entwurf, für die Übergabe der anwendungsspezifischen Transformationen an die Ausgabemodalitäten, vorgestellt.

Die Eingabe-/Ausgabemodalitäten

Die Eingabe-/Ausgabemodalitäten besitzen sowohl die Fähigkeit, die Eingaben des Benutzers, wie auch die Ausgaben der Anwendung, innerhalb eines Kommunikationskanals, verarbeiten zu können.

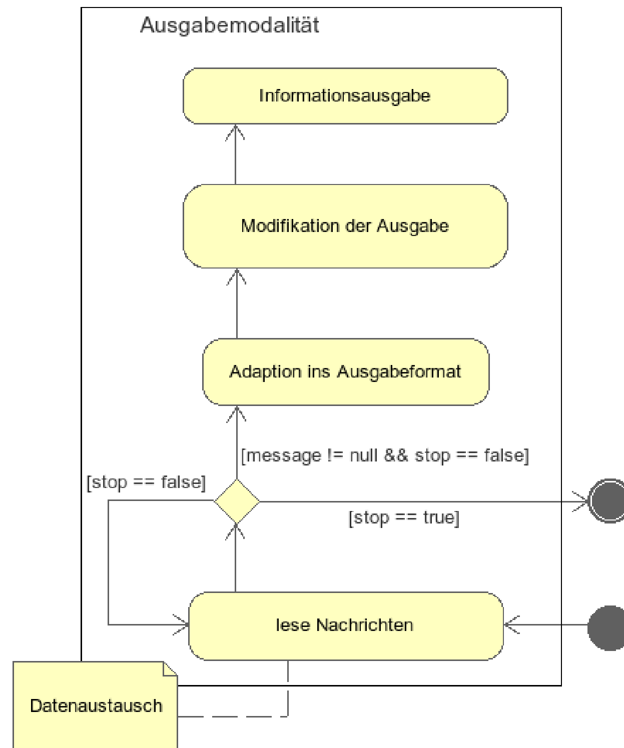


Abbildung 3.4.: Aktiviäten innerhalb der Ausgabemodalität

3.2.2. Die Beschreibung der Modalitäten durch Meta-Informationen

Nach dem Schlüsselkonzept der Sichtbarkeit der SOA (siehe Abschnitt 2.2.2) müssen die Ein- und Ausgabemodalitäten durch eine Servicebeschreibung charakterisiert werden. Diese enthält eine Menge von Informationen über die jeweilige Modalität, die sog. Meta-Informationen. Sie dienen als allgemeine Beschreibung und sind Voraussetzung für das Erkennen und Klassifizieren der vorhandenden Modalitäten zur Laufzeit durch die MSC.

Die Beschreibung der Modalitäten ist selbst plattformunabhängig, um den Einsatz der Modalitäten in heterogenen Umgebungen gewährleisten zu können.

Im EMMA-Format werden Eingabemodalitäten durch die folgenden Attribute (siehe Tabelle 3.1) genauer beschrieben.

Attribut	**mögliche Werte**	**Bedeutung**
`medium`	accoustic | tactile | visual	Unterscheidung des Ein- und Ausgabemediums
`mode`	speech | dtmf_keypad | ink | gui | keys | video | photograph	Unterscheidung der verschiedenen Modi eines Mediums
`function`	recording | transcription | dialog | verification	Unterscheidung der kommunikativen Funktion, z. B. können Spracheingaben aufgenommen werden (recording) oder zur interaktiven Eingabe von Kommandos genutzt werden (dialog)
`verbal`	true | false | 0 | 1	Unterscheidung zwischen verbalen und nonverbalen Eingabe, z. B. müssen handgeschriebene Wörter von symbolischen Handbewegungen wie Kreise oder Linien unterschieden werden können

Tabelle 3.1.: Attribute der Meta-Informationen

Die Klassifizierung der Eingabemodalitäten nach [CDJ+04] kann auch auf die Ausgabemodalitäten erweitert werden. An dieser Stelle muss darauf hingewiesen werden, dass das EMMA-Format nur auf die Eingabemodalitäten begrenzt ist.

Die Beschreibung der Modalitäten erfolgt durch sog. *Java-Properties* Dateien. Die Eingabe- und Ausgabeeigenschaften der Modalitäten werden getrennt voneinander beschrieben. Sie werden durch die Dateien `input.properties` und `output.properties` charakterisiert. Diese enthalten, als Schlüssel/Wert-Paare, die Attribute und dazugehörige Werte aus der Tabelle 3.1.

Für den Fall, dass es sich bei einer Modalität um eine Eingabemodalität handelt, reicht es aus, nur die Eingabeeigenschaften der Modalität zu beschreiben. Das Gleiche gilt für die Ausgabemodalitäten.

Besitzt eine Modalität die Fähigkeit Eingabe- und Ausgabedaten verarbeiten zu können, so müssen die Ein- und Ausgabeeigenschaften durch beide *Java-Properties* Dateien beschrieben werden.

Zusätzlich zu den Attributen der Tabelle 3.1 können weitere herstellerspezifische Daten hinzugefügt werden. Sie können hilfreiche Informationen, für die spätere manuelle Auswahl der Modalitäten durch den Benutzer, enthalten.

Die folgenden Beispiele sollen veranschaulichen, wie die Attribute der Tabelle als Meta-Informationen einer Modalität, im Format der *Java-Properties* Dateien, bereitgestellt werden. Hierbei handelt es sich um eine Ein-/Ausgabemodalität.

Listing 3.3: Meta-Informationen in *input.properties*

```
1 # input properties of modality 'directinput'
2
3 role = modality provider
4 name = DirectInputModality
5 device = mouse,keyboard
6 mode = gui
7 medium = tactile
8 function = dialog
9 verbal = false
```

Listing 3.4: Meta-Informationen in *output.properties*

```
1 # output of modality 'directinput'
2
3 role = modality provider
4 name = DirectInputModality
5 device = display
6 mode = gui
7 medium = visual
8 verbal = false
```

Die *Java-Properties* Dateien bieten den Vorteil, dass sie durch die Java-Klasse `java.util.Properties` unkompliziert eingelesen werden können und dabei automatisch als Speicherrepräsentation zur Verfügung stehen. Die Dateien selbst sind reine Textdateien und unabhängig von der verwendeten Plattform.

3.2.3. Interaktion durch Java Interfaces

Um einen allgemein gültigen Zugriff auf verschiedene Modalitäten gewährleisten zu können, müssen diese eine einheitliche Schnittstelle anbieten. (Abschnitt 2.1.1 und Abschnitt 2.2.2).

Im *OSGi-Framework* wird die Deklaration der Schnittstelle einer Servicekomponente (*Bundle*) durch das Bereitstellen von *Java Interface* Dateien übernommen. Das bedeutet, dass jede Modalität ein allgemein gültiges *Java Interface* implementieren muss, welches die Methoden für den Zugriff auf die Ein- und Ausgabedaten deklariert.

Die folgende Abbildung 3.5 stellt den Entwurf der Schnittstelle der Modalitäten in UML-Notation dar. Die Elemente der Abbildung 3.5 werden nachfolgend erläutert.

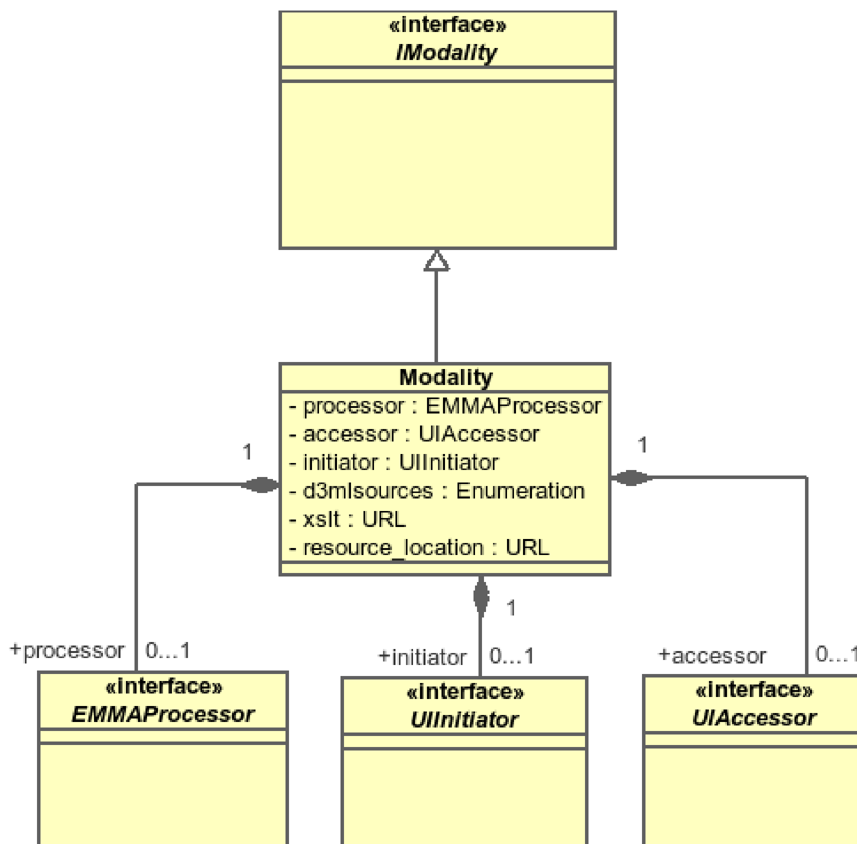


Abbildung 3.5.: Schnittstelle der Modalitäten

IModality

Die Schnittstelle `IModality` ist für den Zugriff auf die aktuelle Modalität zuständig. Sie stellt Methoden bereit, um die Kontrolle über die Eingabe- bzw. Ausgabedaten zu erhalten. Des Weiteren können Informationen von der MSC an die Modalität über die Schnittstelle übergeben werden. Die URL der UI-Beschreibung im D3ML-Format wird bspw. von der MSC an die aktuelle Modalität übergeben. Das heißt, die Schnittstelle der Modalität ist die einzige Möglichkeit des Austauschens von Informationen zwischen den Modalitäten und der MSC.

UIInitiator

Der `UIInitiator` kapselt die Funktion der Initialisierung des *User Interface*. Diese Funktion ist der 1. Schritt der Adaption der abstrakten UI-Beschreibung, im D3ML-Format, an die jeweilige Ausgabe und das dazugehörige Ausgabegerät.

Ein Objekt, welches die Schnittstelle implementiert, sorgt für die Transformation des D3ML-Dokumentes in ein entsprechendes Ausgabeformat. Als Beispiel dient die Transformation von

D3ML in HTML, um die Ausgabeinformationen visuell darzustellen. Die Transformation findet einmalig beim Aktivieren der jeweiligen Ausgabemodalität statt. Der `UIInitiator` ist Bestandteil der Modalität und die MSC kann über das *Interface* `IModality` auf das Objekt des `UIInitator` zugreifen.

UIAccessor

Das Objekt, welches die Schnittstelle des `UIAccessor` implementiert, ist für die Adaption aktueller Informationen an die entsprechende Ausgabe und dem dazugehörigen Ausgabegerät zuständig. Verändert eine EMODE Anwendung die Informationen eines UI-Elementes, so müssen die Veränderungen des Elements an die aktivierte Modalität übergeben werden. Der Zugriff auf die adaptierte Beschnutzerschnittstelle der Modalität erfolgt über die Methoden, die durch dieses *Interface* deklariert werden.

Ein Objekt, welches die Schnittstelle des `UIAccessor` implementiert, ist Bestandteil der aktuellen Modalität und die MSC kann ebenfalls über das Interface `IModality` die Kontrolle über dieses Objekt erhalten.

EMMAProcessor

Damit die MSC die Eingabedaten des Benutzers von der Eingabemodalität an die EMODE Anwendung übergeben kann, braucht sie den Zugriff auf ein Objekt, welches die Schnittstelle des `EMMAProcessor` implementiert. Dieses Objekt hat die Aufgabe die Daten des Eingabegerätes zu interpretieren und das Resultat in das EMMA-Format zu überführen. Es ist Bestandteil der Modalität und die MSC kann die Kontrolle über das Objekt mit Hilfe der Schnittstelle `IModality` erlangen.

3.2.4. Ein- und Ausgabe durch die Sprache

In diesem Abschnitt wird der Entwurf der Spracheingabe- und der Sprachsynthesemodalität vorgestellt. Die Spracheingabemodalität hat die Aufgabe gesprochene Wörter wahrzunehmen und zu verarbeiten. Zur Realisierung der Spracheingabemodalität wird das *Sphinx4-Framework* [WLK+04] verwendet. Der Entwurf der Modalität wird nachfolgend erläutert.

Die Sprachsynthesemodalität hat die Aufgabe die Ausgabeinformationen einer EMODE Anwendung in Textform über die Lautsprecher auszugeben (*Text-to-Speech*). Der Einsatz der Sprachsynthese ist dann sinnvoll, wenn zum Anwendungszeitpunkt kein Display verwendet werden kann. Als Beispiel wird auf das Szenario in der Einführung verwiesen (siehe 1). Die Funktionen der Sprachsynthese werden durch das *FreeTTS-Framework* bereitgestellt.

Realisierung der Spracheingabe mit Hilfe des Sphinx4-Framework

Das *Sphinx4-Framework* ist ein Spracherkennungssytem, welches komplett in Java implementiert wurde und frei verfügbar ist [WLK+04]. Die Erkennung der gesprochenen Wörter erfolgt mit Hilfe vordefinierter Grammatiken. Die Grammatiken bilden die Menge aller wahrnehmbaren Wörter und werden im sog. *Java Speech Grammar Format* (JSGF) definiert. Das Format wird in [Hun00] erläutert.

Eine weitere Spracherkennungsmethode, welche jedoch nicht durch das *Sphinx4-Framework* realisiert wird, kommt gänzlich ohne Grammatiken aus. Alle diktierten Wörter werden frei erkannt.

Die Fehleranfälligkeit ist im Vergleich zu der vorherigen Methode relativ hoch. Die Erkennung der diktierten Wörter benötigt mehr Rechenzeit. Die freie Spracherkennung soll in dieser Arbeit nicht weiter betrachtet werden.

Die grundlegende Funktionsweise der Spracheingabemodalität soll durch das Aktivitätsdiagramm A.3.1 und durch die folgenden Erläuterungen verdeutlicht werden.

Zunächst muss geklärt werden wie die Spracheingabemodalität die Grammatiken im *JSGF*-Format erhält. Jedes D3ML-Dokument enthält die Elemente der multimodalen Benutzerschnittstelle. Jedes Eingabeelement der multimodalen Benutzerschnittstelle enthält als zusätzliches Attribut eine Referenz auf eine *JSGF*-Grammatikdatei. Informationen dazu können in [SAP06] nachgeschlagen werden. In der Grammatikdatei wird die Menge der sprechbaren Wörter definiert, die durch das Eingabeelement erzeugt werden können. Einem Textfeld kann die Grammatikdatei zugeordnet sein, die alle Namen der Anwendungsnutzer enthält. Durch das Sprechen eines Namens, der in der Grammatik enthalten ist, soll er im Textfeld erscheinen und als Eingabe für den Benutzernamen einer Anwendung dienen.

Die Information, die durch das Mikrophon aufgenommen und als *Audio Chunks* bezeichnet werden, werden durch das *Sphinx4*-System nach einem *Hidden Markov Modell* (HMM) untersucht (siehe dazu [WLK+04]). Am Ende der internen Prozesse der *Sphinx4*-Spracherkennung wird die Textrepräsentation der gesprochenen Wörter ausgegeben. Kann das *Sphinx4*-System die Wörter nicht erkennen, so wird kein Text ausgegeben und die Modalität muss den Benutzer um eine erneute Eingabe auffordern.

Anhand der gesprochenen Wörter kann ermittelt werden, welches UI-Element die Eingabe erzeugt hat. Dies ist notwendig, da die Spracheingabemodalität nur eine Eingabequelle besitzt, das Mikrophon. Eine visuelle Eingabemaske hingegen kann mehrere Eingabequellen besitzen, z.B. mehrere Eingabefelder. Die multimodale Benutzerschnittstelle besitzt ebenfalls mehrere Eingabeelemente, denen verschiedene auszulösende Kommandos zugeordnet sind.

Die Ermittlung der UI-Elemente erfolgt mit Hilfe einer Phrase – UI-Element – Abbildung. Sie ordnet jedem sprechbaren Ausdruck die Identifikation des UI-Elementes zu, das auf die Grammatikdatei zeigt, welche den Ausdruck selbst enthält. Nachdem das UI-Element bestimmt wurde, kann auch das auszulösende Kommando ermittelt werden. Das Kommando und dessen Argumente werden, wie in jeder anderen Eingabemodalität, im nächsten Schritt interpretiert (in das EMMA Format) und anschließend, zur Übergabe an den Anwendungscontainer, bereitgestellt.

Sprachsynthese durch das FreeTTS-Framework

Das *FreeTTS (Free Text-to-Speech) Framework* ist ein freies, Java-basiertes Sprachsynthesesystem [WLK05]. Die Modalität der Sprachsynthese erzeugt aus Textdaten gesprochene Wörter mit Hilfe der Lautsprecher. Die Modalität besitzt mehrere Stimmen, die im Klang und im Geschlecht variieren können.

Das *FreeTTS Framework* unterstützt gegenwärtig keine *MarkUp*-Beschreibungen zusammenhängender Ausgabedialoge, wie z.B. SSML, JSML oder VoiceXML. Die Sprachsynthesemodalität besitzt demnach eine Nachrichten-orientierte Funktion, d. h. wenn eine EMODE Anwendung Informationen über die Sprachsynthesemodalität nach außen anbieten möchte, so werden die Datenelemente als einzelne Botschaften versendet und durch die Modalität verarbeitet. Im Szenario einer *Chat*-Anwendung wird der Name eines neuen Kommunikationspartners oder neu eintreffende Nachrichten angesagt.

Die grundlegende Funktionsweise wird in der Abbildung 3.6 veranschaulicht.

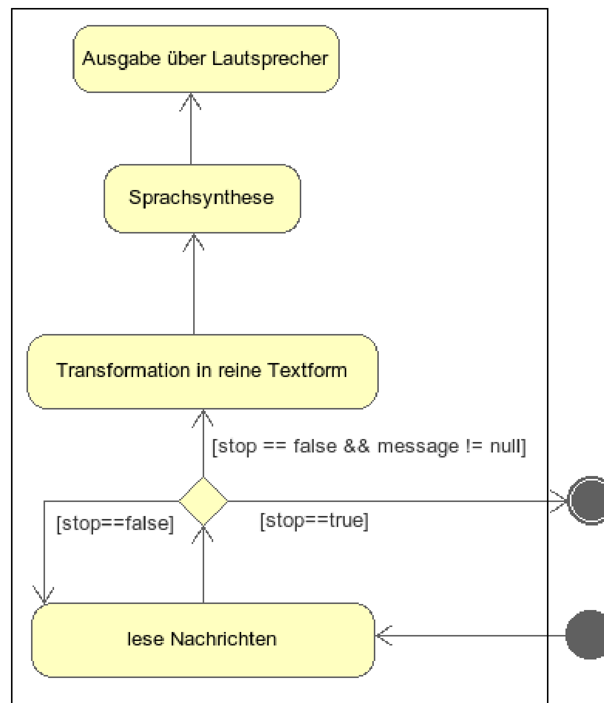


Abbildung 3.6.: Prozesse der Sprachsynthese

3.2.5. Direkte Ein- und visuelle Ausgabe

Der Abschnitt soll die Modalität der direkten Ein- und visuellen Ausgabe vorstellen. Dabei wird hauptsächlich die grundlegende Struktur und Funktionsweise erläutert. Einzelheiten der Implementierung wie z. B. die Betrachtung des Lebenszyklus eines *Applets* innerhalb einer UI-Dialog-Sitzung werden nicht betrachet.

Die Modalität der direkten Ein- und Ausgabe wird mit Hilfe von XHTML realisiert. Es dient der Modalität als Ausgabeformat. Ein Browser fungiert dabei als GUI. Die Modalität entspricht der Kategorie der Ein-/Ausgabemodalität, d. h. sie kann sowohl Eingabeinformationen sowie auch Ausgabeinformationen verarbeiten. Sie ist in der Lage die Eingaben des Benutzers via Browser an die MSC zu versenden und alle Ausgabeinformationen der MSC durch den Browser darstellen zu lassen. Die Funktionen des Senden und Empfangen von Information werden durch ein *Applet* implementiert, welches in dieser Arbeit als sog. *Antenna* bezeichnet wird. Der Entwurf der Modalität soll durch das Klassendiagramm A.3.2 veranschaulicht werden.

Der Browser kann mit Hilfe von speziellen *JavaScript*-Funktionen mit dem *Antenna-Applet* kommunizieren und bestimmte Methoden aufrufen. Eine wichtige Funktion des *Applet* ist das Versenden der Kommandonachrichten an die MSC. Es bildet einen Kommunikationskanal zwischen dem *User Interface (Front End)* und den internen Verarbeitungskomponenten *(Back End)*. Alle *Back End*-Komponenten werden durch ein *OSGi-Bundle* implementiert und stehen der MSC zur Verfügung, darunter befindet sich auch die Komponente der Eingabeinterpretation.

Das folgende Szenario soll die Funktionsweise verdeutlichen.

Ein Benutzer betätigt einen Knopf in der GUI. Dem Knopfelement ist ein bestimmtes Kommando zugeordnet. Die Kommando – UI-Element – Bindung stammt aus der multimodalen Benutzerschnittstelle und wird in der Initialisierungsphase der Modalität in das Ausgabeformat übernommen. Dies wird im Abschnitt 3.5.1 noch genauer erläutert.

Das Kommando und zugehörige Argumente werden, durch die *JavaScript*-Funktionen der Modalität, im XML-Format über das Applet an die *Back End*-Komponenten übergeben. Das Kommando wird anschließend in das EMMA-Format überführt und innerhalb der MSC an den Anwendungscontainer weiter geleitet.

Werden Daten durch den Benutzer, z. B. in ein Textfeld, eingetragen, so kann das UI-Element aufgrund der Veränderung des Zustands und anhand der multimodalen Benutzerschnittstellenbeschreibung, ebenfalls mit Hilfe der *JavaScript*-Funktionen, ein Kommando auslösen. Die Eingabedaten werden in diesem Fall als Kommandoargumente versendet.

Die Ausgabe von neuen Informationen erfolgt, indem das bestehende XHTML-Dokument verändert wird. Es können neue Informationselemente hinzugefügt oder bestehende Elemente verändert bzw. entfernt werden. Genaueres dazu ist in Abschnitt 3.5.2 zu finden. Das *Antenna-Applet* sorgt nach der Veränderung des gegenwärtigen UI-Dialoges für die Aktualisierung der Anzeige im Browser.

3.3. Der Entwurf der EMODE Anwendungen

Eine EMODE Anwendung besteht hauptsächlich aus der Menge der Funktionen der Anwendungslogik. Zum Beispiel besteht eine EMODE *Chat*-Anwendung aus den Funktionen für das Empfangen und Versenden von Nachrichten.

Die EMODE Anwendung soll durch die multimodalen Benutzerschnittstellen kontrolliert werden, d. h. die Auslösung der Funktionen der Anwendung muss durch die verschiedenen Eingaben des Benutzers geschehen. Der Aufruf der Funktionen kann nicht direkt durch die Eingabemodalitäten durchgeführt werden, da die Modalitäten keine Kenntnisse über die Anwendungsfunktionen besitzen. Es existieren die folgenden Varianten für den Aufruf der Anwendungsfunktionen.

Variante 1: Die Eingabemodalitäten versenden vordefinierte Kommandos, in Kombination mit den Eingaben des Benutzers, an die MSC. Die MSC muss die Kommandos aus dem Datenstrom der Benutzereingabe extrahieren und anhand einer Zuordnungstabelle entscheiden, welche Anwendungsfunktion durch welches Kommando ausgeführt werden soll. Dabei muss die MSC die entsprechenden Argumente ebenfalls aus dem Datenstrom der Benutzereingabe ermitteln und sie während des Funktionsaufrufs übergeben.

Variante 2: Die Eingabemodalitäten versenden vordefinierte Kommandos, in Kombination mit den Eingaben des Benutzers, an die MSC. Die MSC leitet den Datenstrom der Benutzereingabe an die Anwendung weiter. Die EMODE Anwendung extrahiert selbst die vordefinierten Kommandos und die zugehörigen Argumente und entscheidet anhand einer internen Zuordnungstabelle, welche Anwendungsfunktion aufgerufen werden soll.

Die 1. Variante vereinfacht die Entwicklung von EMODE Anwendungen. Sie bestehen nach dieser Variante hauptsächlich nur aus den Funktionen der Anwendungslogik. Der Funktionsaufruf wird durch die MSC vorgenommen. Der Entwickler der Anwendung muss der MSC eine Abbildung der vordefinierten Kommandos auf die Menge der Anwendungsfunktionen übergeben. Diese Abbildung muss in einem für die MSC verständlichem Format bereitgestellt werden.

Der Nachteil der 1. Variante ist, dass der Funktionsaufruf durch die MSC nur dann ohne Umstände geschehen kann, wenn die Anwendung in der gleichen Programmiersprache, wie die MSC, implementiert wurde. Die EMODE Laufzeitumgebung soll jedoch soweit wie möglich flexibel sein.

Die 2. Variante überlässt den Aufruf der Anwendungsfunktionen der Anwendung selbst. Die Anwendung muss dafür eine bestimmte Struktur und Funktionen besitzen. Der Anwendungsentwickler muss die Zuordnung der vordefinierten Kommandos an die Anwendung übergeben bzw. integrieren. Der Nachteil dieser Variante ist, dass die Entwicklung einer EMODE Anwendung im Vergleich zur 1. Variante komplexer ist. Der Vorteil, den die 2. Variante bietet, ist, dass die EMODE Anwendung, ohne Rücksicht auf die Programmiersprache, unabhängig von der MSC implementiert werden kann.

In dieser Arbeit soll die 2. Variante realisiert werden, um das Ziel der Flexibilität zu erreichen. Die folgende Abbildung 3.7 zeigt die Grundstruktur einer EMODE Anwendung. Auf Basis der Grundstruktur soll es später möglich sein, standardisierte Komponenten zu verwenden, die die Kommunikation mit der EMODE Laufzeitumgebung bereits implementieren. Ein mögliches Ziel ist dabei die Gestaltung eines *EMODE-Framework* bzw. einer EMODE Funktionsbibliothek.

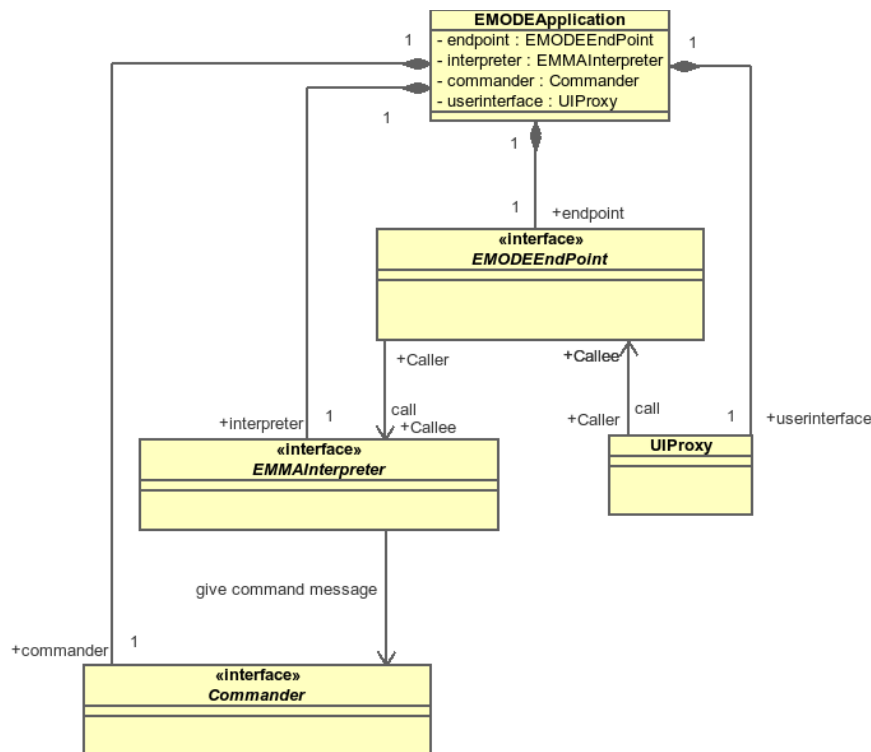


Abbildung 3.7.: die Grundstruktur einer EMODE Anwendung

Die Fähigkeit, zur Kommunikation mit der EMODE Laufzeitumgebung, ist eine wichtige Funktion, die jede EMODE Anwendung besitzen muss. Die Kommunikation beinhaltet das Empfangen der Benutzereingaben und das Versenden von Ausgaben, die durch die MSC verarbeitet und an die multimodalen Benutzerschnittstellen weitergeleitet werden sollen. Demzufolge muss eine EMODE Anwendung eine Komponente besitzen, die die Funktionen der Interprozeßkommunikation implementiert. In Abbildung 3.7 wird diese Komponente als `EMODEEndPoint` dargestellt.

Die Komponente des `EMODEEndPoint` ist für das Empfangen der Benutzereingaben und für das Versenden der Ausgaben zuständig. Im folgenden Abschnitt 3.3.1 wird die Verarbeitung der Benutzereingabe näher erläutert, dabei werden wichtige Komponenten vorgestellt.

Der Prozess der Ausgabeerzeugung und die Verarbeitung der Ausgabedaten wird im darauf folgenden Abschnitt 3.3.2 diskutiert.

3.3.1. Die Verarbeitung der Benutzereingaben durch die EMODE Anwendung

Jede Eingabemodalität interpretiert die Eingaben des Benutzers und sendet die Daten an die MSC. Nach der 2. Variante (siehe Abschnitt 3.3) leitet die MSC die Benutzereingaben an die Applikation weiter. Das bedeutet, die EMODE Anwendung erhält die Benutzereingaben in interpretierter Form, im EMMA Format. Aus diesem Datenstrom der Benutzereingabe muss die Anwendung nun die vordefinierten Kommandos und die dazugehörigen Argumente herauslesen und für die weitere Verarbeitung aufbereiten. Dazu benötigt die EMODE Anwendung eine Komponente, die die Extraktion der Kommandos und weiterer Daten durchführt. Dabei müssen die Kommandos und die Argumente in einem bestimmten Datenmodell vorliegen. Dieses ist anwendungsabbhängig und muss durch den Entwickler vorgegeben werden. In Abbildung 3.7 wird die Extraktionskomponente als `EMMAInterpreter` bezeichnet.

Nach dem die Kommandos und die entsprechenden Argumente aus der Benutzereingabe ermittelt worden, können sie nun für den Aufruf einer Funktion genutzt werden. Der Anwendungsentwickler muss definieren welches Kommando welche Funktion auslöst. Die Zuordnung der Kommandos zu den Funktionen der Anwendung kann fest in die Komponente, die für den Funktionsaufruf zuständig ist, integriert werden, bzw. innerhalb einer Datei beschrieben und dann von der Funktionsaufrufskomponente eingelesen werden. Die Beschreibung der Zuordnung innerhalb einer Datei bietet den Vorteil, dass die Zuordnungen verändert werden können. Dies kann auftreten, wenn die Dialoge der Benutzerschnittstelle verändert werden und neue Kommandos integriert werden sollen. Durch die Zuordnungsdatei kann der Entwickler einem neuen Kommando eine neue Funktion zuordnen, ohne den Quellcode der Anwendung verändern zu müssen. Im Abschnitt 3.3.3.3 wird das Datenmodell der Zuordnungsdatei näher erläutert.

In Abbildung 3.7 wird die Komponente, die für den Aufruf der Funktionen zuständig ist, als `Commander` bezeichnet.

3.3.2. Die Erzeugung der Ausgabedaten durch die EMODE Anwendung

Eine EMODE Anwendung erzeugt im Verlauf der Ausführung verschiedene Ausgaben, sie besitzt jedoch keinen direkten Zugang zu den multimodalen Benutzerschnittstellen. Um die verschiedenen Informationen ausgeben zu können, muss die EMODE Anwendung die gewünschten Ausgabedienste kontaktieren. Der Kontakt wird durch die MSC hergestellt. Die EMODE Anwendung gibt der MSC durch eine Beschreibung von Anforderungseigenschaften bekannt, welche Ausgabedienste sie nutzen möchte. Im Abschnitt 3.3.3.1 wird auf die Beschreibung der Anforderungen genauer eingegangen.

Nach dem die EMODE Anwendung durch die MSC die jeweilige Ausgabemodalität nutzen kann, versendet sie die Ausgabeinformationen. Das Versenden der Ausgabe wird mit Hilfe der Komponente durchgeführt, die für die Interprozesskommunikation zuständig ist, der `EMODEEndPoint`. Die MSC empfängt die Ausgabeinformationen und gibt sie an die entsprechenden Ausgabemodalitäten weiter. Dabei finden zusätzliche Verarbeitungsschritte statt, die im Abschnitt 3.5 diskutiert werden. Die Ausgabeinformationen sind stark abhängig von der jeweiligen Situation der Anwendung. Als Beispiel dient das folgende Anwendungsszenario.

In einer *Chat*-Anwendung können sich im Verlauf der Anwendung neue Kommunikationspartner anmelden oder abmelden. Während der Anmeldung wird der Name des Kommunikationspartners in eine Liste hinzugefügt. Meldet sich ein Kommunikationspartner ab, so wird sein Name aus der Liste entfernt. Diese Liste kann visuell dargestellt werden und dient dem Nutzer als Überblick der aktuellen Kommunikationspartner. Bei jeder Veränderung der Liste der Kommunikationspartner muss auch die visuelle Darstellung aktualisiert werden. Das bedeutet die Anwendung muss die Veränderungen, mit Hilfe von Nachrichten an die Ausgabe, übergeben. In der Situation

des Anmeldens eines neuen Kommunikationspartners XY beinhaltet die Ausgabenachricht die Information „Füge einen neuen Eintrag XY in die Liste hinzu“. Das Gleiche gilt für die Situation des Abmeldens eines Kommunikationspartners XY. Die Ausgabenachricht lautet in diesem Fall „Entferne den Eintrag XY aus der Liste der Kommunikationspartner“.

Das Beispiel zeigt das in den unterschiedlichen Situationen verschiedene Informationen an die Ausgabe gesendet werden. Im Fall der EMODE Anwendungen bedeutet es, dass die Ausgabenachrichten an die jeweiligen Ausgabemodalitäten übergeben werden.

Des Weiteren müssen die Ausgabeinformationen in einem speziellen Datenmodell vorliegen, damit sie von der jeweiligen Ausgabemodalität verarbeitet werden können. Dabei muss beachtet werden, dass die verschiedenen Ausgabemodalitäten völlig unabhängig von der EMODE Anwendung existieren, daher sollte das Datenmodell der Ausgabenachrichten soweit wie möglich unabhängig von der jeweiligen Ausgabemodalität sein, um somit die Nutzung mehrerer Ausgabemodalitäten zu ermöglichen. Die MSC ist dafür zuständig die Ausgabenachrichten an die Ausgabemodalitäten anzupassen. Der Prozess der Anpassung (Adaption) wird im Abschnitt 3.5 genauer beschrieben.

Die EMODE Anwendung benötigt eine Komponente, welche die Aufgabe der Erzeugung der verschiedenen Ausgabenachrichten, in der jeweiligen Anwendungssituation, übernimmt. Diese Komponente ist ebenfalls für die Übermittlung der Ausgabeinformationen an die MSC zuständig. Sie fungiert aus Sicht der Anwendung als Stellvertreter für die Benutzerschnittstelle. Die Komponente wird in der Abbildung 3.7 als `UIProxy` bezeichnet.

3.3.3. Die Meta-Informationen der EMODE Anwendungen

Eine multimodale Anwendung, die möglicherweise auf einem mobilen Endgerät ausgeführt werden soll, besitzt die Eigenschaft eines dynamisch variierenden Kontextes. Das Beispiel in der Einführung 1 zeigt, das die unterschiedlichen Umweltbedingungen den Kontext der Anwendung, z.B. die Verfügbarkeit bestimmter Ein- und Ausgabegeräte, verändern können. Nicht nur mobile Endgeräte, sondern auch Desktop-Rechner variieren in Bezug auf die angeschlossenen Ein- und Ausgabegeräte. Soll eine Anwendung auf einen anderen Rechner transferiert werden, so ändert sich aus der Sicht der Anwendung die Verfügbarkeit der Ein- und Ausgabegeräte. Für den Anwendungsentwickler bedeutet dies, dass er nicht mehr die Verfügbarkeit der verschiedenen Ein- und Ausgabegeräte voraussetzen kann. Es kommt zu einem Wechsel des Entwicklungsparadigma, die Anwendung implementiert selbst keine Ein- und Ausgabefunktionen mehr, sondern sie beschreibt deklarativ welche Ein- und Ausgabegeräte mit welchen Eigenschaften genutzt werden sollen. Dazu muss die Anwendung die verschiedenen Beschreibungen, die Meta-Informationen der Anwendung, nach außen anbieten.

Im folgenden Abschnitt 3.3.3.1 wird das Datenmodell für die Beschreibungen der Eigenschaften der zu verwendenden Ein- und Ausgabemodalitäten vorgestellt.

Der Benutzer einer herkömmlichen Anwendung löst bestimmte Funktionen direkt durch die verschiedenen Eingabeelement aus. Eine EMODE Anwendung besitzt jedoch keine direkte Ankopplung an die Eingabeelemente der Benutzerschnittstelle. Die Eingabedaten werden innerhalb eines standardisierten Formates, in dieser Arbeit ist es das EMMA Format, interpretiert und an die Anwendung übergeben. Dabei werden keine direkten Funktionsaufrufe durchgeführt. Der Anwendungsentwickler muss an die Elemente der multimodalen Benutzerschnittstelle unterschiedliche Kommandos koppeln, die bei der Betätigung des jeweiligen Elementes in die Interpretation der Eingabedaten integriert und mit ihnen an die Anwendung übergeben werden. Die Anwendung muss die Kommandos aus dem Datenstrom ermitteln und eine bestimmte Funktion aufrufen. Die

genaue Erläuterung ist im Abschnitt 3.4 enthalten. Die Voraussetzung dabei ist, dass die Abbildung der Kommandos auf die Funktionen durch den Anwendungsentwickler bereitgestellt wird. Der folgende Abschnitt 3.3.3.3 beschreibt das Datenmodell, für die Abbildung der Kommandos auf die Menge der Anwendungsfunktionen.

Die Kommandos werden durch den Anwendungsentwickler innerhalb der Beschreibung der multimodalen Benutzerschnittstelle definiert. Die Erläuterung dazu sind im Abschnitt 3.3.3.2 zu finden.

3.3.3.1. Die Beschreibung der Anforderung an die Modalitäten

Die MSC ist ein Vermittler zwischen den bereitstehenden Ein- und Ausgabediensten, den Modalitäten, und den Dienstnutzern, den EMODE Anwendungen. Sie hat aufgrund der Meta-Informationen der Modalitäten Kenntnis über deren Eigenschaften. Des Weiteren benötigt sie Informationen darüber, welche Eigenschaften eine Modalität haben muss, damit sie der jeweiligen EMODE Anwendung zugeordnet werden kann. Dies bedeutet, die MSC muss von der Anwendung eine Beschreibung über die geforderten Modalitäten erhalten.

Die Beschreibung der geforderten Modalitäten soll analog zur Beschreibung der Meta-Informationen der Modalitäten (Abschnitt 3.2) innerhalb von *Java-Properties* Dateien erfolgen. Die Eingabeeigenschaften der zu verwendenden Eingabemodalitäten werden in der Datei *input.properties* beschrieben. Die Ausgabeeigenschaften der zu verwendenden Ausgabemodalitäten werden in der Datei *output.properties* beschrieben. Als Beispiel dient das folgende Listing der *input.properties* und *output.properties* Dateien einer multimodalen *Chat*-Anwendung.

Listing 3.5: Meta-Informationen in *input.properties*

```
1 # wanted input modality properties
2
3 device = mouse, keyboard
4 mode = gui
5 function = dialog
6 medium = tactile
```

Listing 3.6: Meta-Informationen in *output.properties*

```
1 # wanted output modality properties
2
3 device = display, speaker
4 mode = gui, speech
5 function = dialog
6 medium = visual, acoustic
7 context = elements2html.xsl -> gui, elements2vxml.xsl -> speech
```

Zur Beschreibung der geforderten Eigenschaften der zu verwendenden Modalitäten sollen die Attribute aus der Tabelle 3.1 aus Abschnitt 3.2 verwendet werden.

Eine EMODE Anwendung soll mehrere Modalitäten verwenden können. Die Dateien zur Beschreibung der Ein- und Ausgabeeigenschaften umfassen demzufolge eine Menge von Ein- und Ausgabemodalitäten. In der jeweiligen Datei beschreibt ein Attribut eine Menge von möglichen Eigenschaften innerhalb des entsprechenden Wertebereichs. Im Beispiel der Datei *output.properties* definiert das Attribut „device“ die Menge aller nutzbaren Ausgabgeräte. Die Wertemengen der Attribute werden in den Dateien *input.properties* und *output.properties* als Listen von Einträgen, die durch Kommata voneinander getrennt werden (*comma separated list*), dargestellt.

Anhand der Beschreibung der geforderten Eigenschaften kann die MSC aus der Menge der verfügbaren Modalitäten die nutzbaren Modalitäten herausfiltern. Die MSC muss während des Suchvorgangs überprüfen, ob die Eigenschaften der aktuellen Modalität in der Menge der geforderten Eigenschaften enthalten sind. Im positiven Fall wird die aktuelle Modalität in die Menge der nutzbaren Modalitäten der jeweiligen Anwendung hinzugefügt und aus der Menge der verfügbaren Modalitäten entfernt. Die Modalität steht im weiteren Verlauf nur noch der aktuellen Anwendung zur Verfügung. Der Mechanismus der Verbindung einer Modalität mit einer EMODE Anwendung wird in Abschnitt 4.2 näher erläutert.

Das zusätzliche Attribut `context` (siehe Lising 3.6) enthält keine Informationen darüber, welche Eigenschaft eine Ausgabemodalität haben muss, sondern es enthält Informationen über die Zuordnung von anwendungsspezifischen Transformationen zu einer Ausgabemodalität. Die Problematik der anwendungsspezifischen Transformationen wurde am Ende des Abschnitts 3.1.2.2 bereits geschildert. Mit Hilfe des `context`-Attributs sollen die Transformationen der EMODE Anwendung an die jeweilige Ausgabemodalität übergeben werden.

Der Anwendungsentwickler muss wissen, welches Ausgabeformat durch die Ausgabemodalität verwendet wird, um die Transformationen korrekt durch *XSLT*-Dateien beschreiben zu können. Der Dateiname der jeweiligen Transformationsbeschreibung muss exakt im `contex`-Attribut der *output.properties*-Datei enthalten sein. Dabei wird vorausgesetzt, dass die Transformation im gleichen Verzeichnis wie die Datei *output.properties* enthalten ist.

Die anwendungsspezifische Transformation wird durch die Angabe eines Ausgabeattribut indirekt an eine entsprechende Ausgabemodalität gebunden. Im Beispiel 3.6 wird die Transformation an die Modalität gebunden, welche im Attribut `mode` den Wert „gui“ besitzt. Der Anwendungsentwickler setzt dabei voraus, dass die GUI-Ausgabemodalität HTML als Ausgabeformat verwendet. Nachdem die anwendungsspezifischen Transformationen an die jeweilige Ausgabemodalität übergeben wurden, können sie zur Anpassung der Ausgabeinformationen der EMODE Anwendung genutzt werden. Der Mechanismus, zur Anpassung der Ausgabeinformationen der EMODE Anwendung an das Ausgabeformat der Modalität, wird in 3.5 noch genauer erläutert.

3.3.3.2. Kommandodefinitionen in der multimodalen Benutzerschnittstelle

Im Abschnitt 3.3 (siehe Variante 2) wurde beschrieben, dass für den Aufruf der Anwendungsfunktionen verschiedene Kommandos an die unterschiedlichen Eingabeelemente der Benutzerschnittstelle gekoppelt werden müssen.

In 2.4.1.2 und 2.4.1.3 ist die Struktur der multimodalen Benutzerschnittstelle beschrieben. Mit Hilfe der XForms-Spezifikation werden die Elemente der Benutzerschnittstelle definiert und an das Datenmodell der Anwendung gebunden. Die Kopplung der Kommandos an die Elemente der Benutzerschnittstelle erfolgt ebenfalls innerhalb der Beschreibung des multimodalen *User Interface*, den D3ML-Dokumenten. Die Beschreibungen enthalten die abstrakte Struktur der multimodalen Benutzerschnittstelle.

Um ein Kommando an ein Element der Benutzerschnittstelle zu binden, kann der Entwickler aus den folgenden 2 Möglichkeiten wählen.

Die 1. Möglichkeit besteht darin, das Kommandoelement **innerhalb** des UI-Elementes zu notieren. Das folgende Beispiel 3.7 soll diese Variante verdeutlichen.

Listing 3.7: Bindung des Kommando 'SETNICKNAME' an ein Trigger-Element

```
1 ...
2 <xforms:input id="in1" bind="b2">
3   <xforms:label>Your Nickname please:</xforms:label>
```

```
4  </xforms:input>
5  ...
6  <xforms:trigger id="tr1" ev:event="DOMActivate">
7    <xforms:label>accept</xforms:label>
8    <!-- triggers command -->
9    <emode:command id="c1" name="SETNICKNAME">
10     <command>
11       <action>SETNICKNAME</action>
12       <message ref="in1"/>
13     </command>
14   </emode:command>
15 </xforms:trigger>
```

Im Beispiel 3.7 wird an das UI-Element `<xforms:trigger>` das Kommando „SETNICKNAME“ gebunden. Die Bindung wird durch das Einbetten des Elementes `<emode:command>` in das Element des *User Interface* hergestellt. Das Element `<emode:command>` definiert die Nachricht, die in den Datenstrom der Eingabe der jeweiligen Modalität integriert werden soll.

Die 2.Möglichkeit besteht darin, das Kommandoelement mit Hilfe von **XML Events** (siehe [MPR03]) an das UI-Element zu binden. Diese Variante wird im Beispiel 3.8 angewandt.

Listing 3.8: Bindung des Kommando 'SETSELECTION' an ein select-Element

```
1  ...
2  <ev:listener observer="sel1" event="xforms-select" handler="#cSelect"/>
3  <emode:command id="cSelect" name="SETSELECTION">
4    <xforms:insert context="instance('i_clientselection')"
5     nodeset="selected-clients" origin="currentSelection('sel1')"/>
6  </emode:command>
7  ...
8  <xforms:select id="sel1" model="m_clientselection"
9   ref="selected-clients">
10   <!--referenced model contains selected items  -->
11   <xforms:label>Client List:</xforms:label>
12   <xforms:itemset bind="b_clist" ev:event="xforms-select">
13   <!--referenced model contains available items-->
14    <xforms:label ref="client-name" />
15    <xforms:value ref="client-id" />
16   </xforms:itemset>
17 </xforms:select>
```

Im Beispiel 3.8 besitzt das Element `<xforms:itemset>` das *XML-Event*-Attribut `ev:event`. Es dient der Kennzeichnung des Typs des Ereignisses, welches durch das Element ausgelöst werden soll. Das Element `<xforms:itemset>` beschreibt in Kombination mit dem Elternelement `<xforms:select>` eine dynamisch veränderbare Auswahlliste. Wird ein Eintrag dieser Auswahlliste gewählt, so soll das Ereignis `xforms-select` ausgelöst werden. Das Element `<ev:listener>` besitzt die Aufgabe, eine Behandlungsroutine an ein vorkommendes Ereignis zu binden. Die Bindung wird durch spezielle Attribute des `<ev:listener>` Elementes genauer definiert. Ein solches Attribut ist das `observer`-Attribut, welches auf das Element der Benutzerschnittstelle bzw. auf ein Elternelement zeigt. Im Beispiel zeigt es auf das `<xforms:select>`-Element, da dies das Element der Benutzerschnittstelle ist.

Im Szenario des Beispiels soll während der Auswahl eines Eintrags das Element `<xforms:insert>` in den Datenstrom der Eingabe integriert werden.

Die Verwendung von `<ev:listener>` Elementen ist dann sinnvoll wenn bestimmte Elemente der Benutzerschnittstelle keine Ereignisse bzw. Aktionen auslösen können. Besonders relevant ist dies in Anbetracht möglicher Ein- und Ausgabemodalitäten.

Die Kopplung der Kommandos mit den Eingabeelementen der Benutzerschnittstelle, muss durch die Eingabemodalität übernommen werden. Dies erfolgt anhand modalitätenspezifischer Transformationen.

Des Weiteren ist die Eingabemodalität für die Übermittlung und für die Interpretation der Eingabedaten zuständig, dabei müssen die Kommandoelemente in den Datenstrom der Eingabe integriert werden.

3.3.3.3. Applikationsfunktionen und Kommandos

Nach dem Entwurf der EMODE Anwendungen (siehe Abschnitt 3.3) sollen die verschiedene Applikationsfunktionen durch Kommandos ausgelöst werden. Die Kommandoinformationen werden durch die Eingabemodalitäten in den Strom der Benutzereingaben integriert. Die EMODE Anwendung extrahiert die eingebetteten Kommandos und weitere Eingabeinformationen, und führt anhand dieser den Funktionsaufruf durch.

Im Prozess des Funktionsaufrufs werden die empfangenen Kommandos und Argumente anhand einer Zuordnungsvorschrift auf die Menge der Applikationsfunktionen abgebildet. Die Zuordnung soll mit Hilfe einer speziellen Datei beschrieben werden. Die EMODE Anwendung liest die Beschreibungsdatei ein und erzeugt intern eine Abbildungstabelle. Sie dient der Ermittlung der aufzurufenden Methode/Funktion der Anwendung.

Das folgende Listing 3.9 zeigt die Beschreibung der Abbildung eines Kommandos auf eine Anwendungsfunktion. Das Listing B.5.1 beinhaltet die zugehörige *XMLSchema*-Datei.

Listing 3.9: Abbildung des Kommando 'SEND' auf Anwendungsfunktion 'sendMsg'

```
<emode:binding
 xmlns:emode="http://www.inf.tu-dresden.de/2007/EMODE/cmd2function">
  ...
  <emode:binding-item>
    <emode:command>
      <emode:action>SEND</emode:action>
      <client-name id="arg1"/>
      <client-id id="arg2"/>
      <message id="arg3"/>
    </emode:command>
    <emode:function>
      <emode:name>sendMsg</emode:name>
      <emode:argumentlist>
        <client-name id="arg1"/>
        <client-id id="arg2"/>
        <message id="arg3"/>
      </emode:argumentlist>
    </emode:function>
  </emode:binding-item>
  ...
</emode:binding>
```

Die Abbildungen der verschiedenen Kommandos auf die Menge der Anwendungsfunktionen erfolgt innerhalb des Wurzelelementes `<emode:binding>`. Sie unterliegen der Bedingung, dass einem Kommando nur eine Funktion zugeordnet werden kann, um somit Zweideutigkeiten zu vermeiden.

Eine Abbildung wird durch das Element `<emode:binding-item>` beschrieben. Es setzt sich aus den Kindelementen `<emode:command>` und `<emode:function>` zusammen. Im Beispiel wird das

Kommando „SEND“ auf die Anwendungsmethode mit dem Namen „sendMsg“ abgebildet. Innerhalb von `<emode:command>` wird das Kommando und die dazugehörigen Argumente charakterisiert. Im Element `<emode:function>` wird die Funktion der Anwendung und die notwendige Argumentenliste beschrieben. Der Name der Funktion wird durch das Kindelement `<emode:name>` und die Argumentenliste durch `<emode:argumentlist>` angegeben.

Damit die Methode der Anwendung erfolgreich aufgerufen werden kann, müssen alle Elemente in `<emode:argumentlist>`, entsprechend der Methodensignatur im Quellcode, geordnet sein. Die Ordnung muss durch den Anwendungsentwickler vorgegeben werden.

Die Argumente eines Kommandos lassen sich nur anhand der angegebenen Namen identifizieren, d. h. die Argumentenelemente müssen exakt der Kommandodefinition, innerhalb des jeweiligen D3ML-Dokumentes, entsprechen.

Das folgende Listing 3.10 zeigt die Benutzereingabe in interpretierter Form. Es soll die Notwendigkeit der Gleichheit der Kommandoargumente und der Funktionsargumente aus dem vorherigen Beispiel verdeutlichen.

Listing 3.10: Kommando 'SEND' in der Benutzereingabe

```
1  ...
2  <emma:emma version="1.0"
3   xmlns:emma="http://www.w3.org/2003/04/emma">
4    <emma:interpretation id="ip0"
5     medium="tactile"
6     mode="gui"
7     verbal="false"
8     function="dialog"
9     start="1179266947046">
10     <command>
11       <action>SEND</action>
12       <client-name>Carsten</client-name>
13       <client-id>2</client-id>
14       <message>hello</message>
15     </command>
16   </emma:interpretation>
17 </emma:emma>
18 ...
```

Die Kommandonachrichten gelangen von der Eingabemodalität über die MSC zur Anwendung. Im vorherigen Beispiel besteht die Kommandonachricht aus den Elementen, die in `<command>` enthalten sind. Die Reihenfolge der Argumente kann innerhalb des Kommandos beliebig sein, es muss jedoch darauf geachtet werden, dass die Namen der Argumentenelemente mit denen, innerhalb der Zurordnungsdatei, übereinstimmen.

Die Zuordnungsdatei dient der Ermittlung der Methodensignatur der jeweiligen Anwendungsfunktion. Mit Hilfe der Methodensignatur und den extrahierten Argumentwerten kann die EMODE Anwendung, durch den Reflektionsmechanismus der verwendeten Programmiersprache, die zugehörige Methode/Funktion aufrufen.

3.4. Die Interpretation der Eingabe

Bisher wurde die Verarbeitung der Eingabedaten in den verschiedenen Abschnitten bereits diskutiert, bzw. teilweise näher erläutert. In diesem Abschnitt wird der Prozess der Eingabeverarbeitung und die dazugehörigen Teilkomponenten in der Gesamtheit diskutiert.

3.4.1. Die Eingabeverarbeitung innerhalb der Eingabemodalität

In 3.2.1 werden die internen Prozesse der Eingabemodalität durch die Abbildung 3.3 veranschaulicht. Die grundlegenden Verarbeitungsschritte werden nachfolgend erläutert.

Die Eingabe von Daten durch den Benutzer

Die Beschreibung der multimodalen Benutzerschnittstelle wird nach Abschnitt 3.1.2 an alle zugeordneten Modalitäten übergeben. Für die Eingabemodalitäten sind nur die Eingabeelemente der Benutzerschnittstelle und die dazugehörigen Daten- und Aktionsbeschreibungen relevant. Die Bindung von Eingabeelement und Kommando muss durch die Eingabemodalität aufgelöst werden, sie muss ermitteln welches Element welches Kommando ausgibt. Während der Auflösung der Bindungen kann die Eingabemodalität optionale modalitätenspezifische Operationen ausführen. Eine visuelle Eingabemodalität, die auf Basis von HTML operiert, nutzt verschiedene *JavaScript*-Funktionen, um die Elemente innerhalb des *Document Object Model* (DOM) des HTML-Dokumentes aufzuspüren und um deren aktuelle Werte auszulesen. Des Weiteren können zusätzliche Vorverarbeitungen durchgeführt werden, um die Beschreibung der Aktionen mit den aktuellen Eingabedaten zu versehen, z.B. werden die Eingabedaten durch die *JavaScript*-Funktionen in eine XML-Beschreibung überführt.

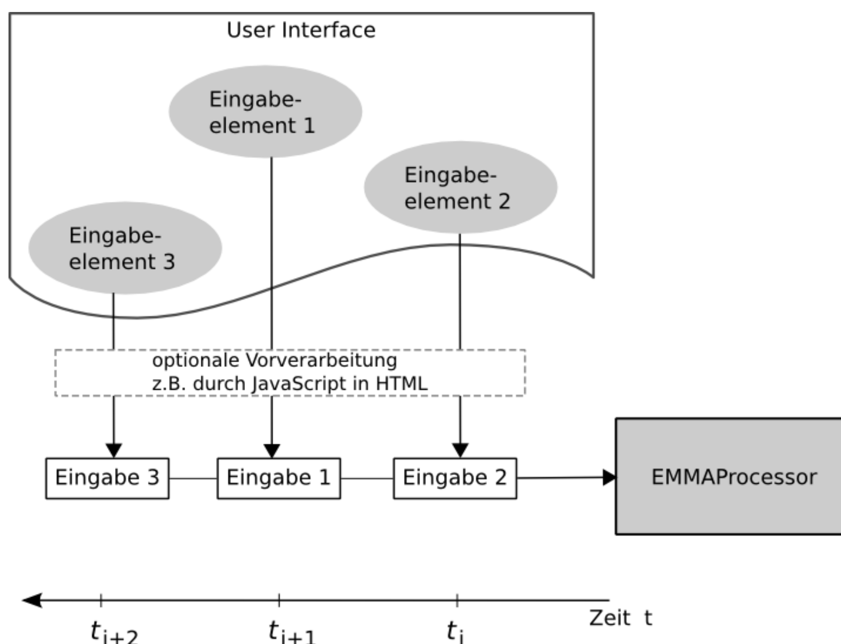


Abbildung 3.8.: Eingabedatenstrom

Jedes Eingabeelement erzeugt einen eigenen Datenstrom und sendet diesen an das Objekt der Klasse `EMMAProcessor` (siehe Abb. 3.8). Er erhält die Eingabedaten in serieller Form. Das Objekt hat die Aufgabe, die verschiedenen Eingaben des Benutzers zu interpretieren. Ein Teil dieser Aufgabe wird bereits durch den Anwendungsentwickler durchgeführt. In der multimodalen Benutzerschnittstelle, im D3ML-Format, werden die Eingabeelemente und die dazugehörigen Kommandos miteinander verbunden. Das Element `<emode:command>` gibt dabei an, welche XML-Elemente in den Datenstrom der Eingabe integriert werden sollen (siehe Listing 3.7). Um die Einbettung der XML-Elemente muss sich die jeweilige Eingabemodalität kümmern.

Die Interpretation der Benutzerdaten im EMMA Format

Die Eingaben des Benutzers werden mit den Kommandoelementen verflochten und an den `EMMAProcessor` übergeben. Er führt folgende Verarbeitungsschritte durch.

1. Analyse der Eingabedaten
2. Überführung der Eingabedaten in das EMMA-Format

Die Analyse der Eingabedaten ist notwendig, um mögliche interne Referenzen aufzulösen. Das Listing 3.7 im Abschnitt 3.3.3.2 enthält innerhalb des `<emode:command>` Elementes in `<message>` eine Referenz auf das Eingabeelement mit der Identifikation „in1“ . Der Inhalt des referenzierten Eingabeelementes muss in `<message>` eingefügt werden.

Die Überführung in das EMMA-Format erfolgt durch die Einbettung der Eingabedaten in das EMMA-Element `<emma:interpretation>`. Es wird zusätzlich mit Attributen versehen, die in der Tabelle 3.1 in Abschnitt 3.2.2 vorgestellt wurden. Des Weiteren wird die EMMA-Interpretation mit einem Zeitstempel versehen. Dieser ist im `start`-Attribut enthalten. Das Listing 3.10 soll dabei als Beispiel dienen.

Das Versenden der interpretierten Eingabedaten

Nachdem der `EMMAProcessor` alle Verarbeitungsschritte durchgeführt hat, übergibt er die interpretierten Eingabedaten an den dazugehörigen EMODE Anwendungscontainer.

3.4.2. Die Eingabeverarbeitung innerhalb des Anwendungscontainer

Ist eine Eingabemodalität durch die MSC an eine Anwendung gebunden worden, so kann der Anwendungscontainer alle interpretierten Eingabedaten von der Eingabemodalität abrufen und durch das Objekt des `EMODEMessageInputProcessor` analysieren lassen (siehe A.2). Es sucht im Datenstrom der interpretierten Eingabe nach bestimmten Textelementen, sog. *Markup*-Aktionen. Die nachfolgende Tabelle 3.2 soll einen Überblick über die *Markup*-Elemente geben.

Das folgende Diagramm (Abb. 3.9) stellt die einzelnen Prozesse der Verarbeitung der Benutzereingabe und deren Auswirkung auf die multimodale Benutzerschnittstelle und den *User Interface*'s der Modalitäten dar. Die einzelnen Schritte werden durch die Nummerierung gekennzeichnet und anschließend erläutert.

(1) Der Anwendungscontainer übergibt, in der Initialisierungsphase der Modalitäten, die Referenzen der einzelnen D3ML-Dokumente in URL-Form. Mit Hilfe der URL der D3ML-Dokumente, kann die Modalität auf die Beschreibung der multimodalen Benutzerschnittstelle zugreifen.

(2) Die Eingabemodalitäten suchen in den D3ML-Dokumenten nach Eingabeelementen und deren zugehörige Kommando- und Aktionenbeschreibungen. Innerhalb der Aktionenbeschreibungen können Referenzen auf andere Eingabeelemente enthalten sein. Die Daten der referenzierten Eingabeelemente werden in die Aktionenbeschreibungen eingefügt. Nachdem alle Eingabeinformationen bereitstehen, können sie interpretiert werden, d. h. die Eingabemodalität überführt die Eingabedaten in das EMMA-Format [CDJ+04].

Aktionselement	<xforms:setvalue ref=“instance(id)“ nodeset=“/to/instance/node“ value=“Wert“/>
Beschreibung	Das Aktionselement trägt den Wert *value* in das Datenelement *nodeset* der XForms-Dateninstanz *ref*, mit der Identifikation *id*, ein (siehe [Boy07]).
Aktionselement	<xforms:insert context=“instance(id)“ nodeset=“/copy/to/node/“ origin=“/copy/this/node“/>
Beschreibung	Das Aktionselement kopiert das Datenelement *origin* in das Kindelement *nodeset*, der XForms-Dateninstanz *context*, mit der Identifikation *id* (siehe [Boy07]).
Aktionselement	<emode:value-of select=“/xpath/to/node/value“/>
Beschreibung	Das Aktionselement dient dem Abfragen der Informationen des Datenelementes, welches durch das *select*-Attribut angezeigt wird. Es gibt das Ergebnis der Abfrage zurück.
Aktionselement	<emode:open-url url=“nextdocument.d3ml“/>
Beschreibung	Das Aktionselement dient dem Aufruf des nächsten D3ML Dokumentes. Es zeigt der MSC und der EMODE Anwendung an, wann und welcher neue Dialog geöffnet werden soll. Das *url*-Attribute gibt den Namen der D3ML-Datei an.

Tabelle 3.2.: mögliche Aktionselemente in der Beschreibung der multimodalen Benutzerschnittstelle

(3) Nachdem die Eingabemodalität alle aktuellen Eingabedaten interpretiert hat, werden sie für den Anwendungscontainer bereitgestellt. Dieser ist für das Einlesen des Datenstroms verantwortlich. Die Kommunikation erfolgt durch ein *Pipe*-ähnliches Verfahren, dabei schreibt die Eingabemodalität auf der einen Seite in einen Ausgabestrom (*PipedOutputStream*). Der Anwendungscontainer liest auf der anderen Seite, von einem Eingabestrom (*PipedInputStream*), alle interpretierten Daten der jeweiligen Eingabemodalität ein.

(4) Die Eingabedaten werden in diesem Teilschritt nach speziellen *Markup*-Elementen untersucht. Die Elemente stellen Aktionsbeschreibungen dar und transportieren Informationen. Es kann sich dabei um das Hinzufügen von neuen Daten in die multimodale Benutzerschnittstelle oder um das Abfragen von bereits enthaltenen Informationen handeln (siehe Tabelle 3.2).

(5.1) Wenn neue Daten in die multimodale Benutzerschnittstelle integriert werden, so muss sich dies auch in der Ausgabe niederschlagen. Das bedeutet alle neuen Informationselemente müssen an alle verfügbaren Ausgabemodalitäten übermittelt werden.

(5.2) Alle Kommandoelemente und dazugehörige Argumente, die im Strom der Eingabedaten

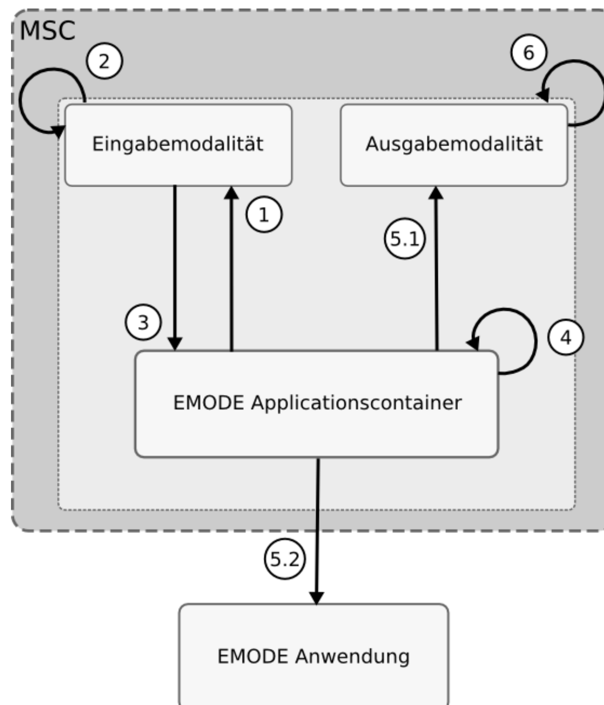


Abbildung 3.9.: Verarbeitung der Eingabe im EMODEApplicationContainer

enthalten sind, werden an die EMODE Anwendung, durch die Mechanismen der Interprozeßkommunikation, übergeben.

(6) Die Ausgabemodalitäten fügen neue Informationselemente in das *User Interface* ein, bzw. verändern die eigene Ausgabe so, dass sie sich in einem äquivalenten Zustand zur multimodalen Benutzerschnittstelle befinden.

Werden der multimodalen Benutzerschnittstelle neue Informationselemente hinzugefügt oder bestehende Elemente verändert bzw. entfernt, so werden die einzelnen Veränderungen, auf Basis einzelner Modifikationsanweisungen, auch im *User Interface* der jeweiligen Modalität durchgeführt. Ein alternatives Verfahren besteht darin, dass nur die multimodale Benutzerschnittstelle durch die Benutzereingaben oder Anwendungsausgaben modifiziert wird. Nachdem das geschehen ist, kann der Anwendungscontainer die Ausgabemodalitäten dazu auffordern, die modifizierten D3ML Dokumente erneut in das Ausgabeformat zu transformieren und die Darstellung zu aktualisieren.

Der Nachteil dieses Verfahrens besteht darin, dass der XSLT-Prozessor die betroffene D3ML-Datei und die XSLT-Datei komplett neu einlesen und transformieren muss. Im Vergleich zur Variante der Modifikation einzelner UI-Elemente ist dieses Verfahren langsamer und aufwendiger. Dementsprechend soll die vorherige Alternative in dieser Arbeit realisiert werden.

3.5. Adaption der Benutzerschnittstelle

In diesem Abschnitt soll der Prozess der Adaption der multimodalen Benutzerschnittstelle am Beispiel der visuellen Ausgabemodalität erläutert werden. Sie verwendet einen *HTML-Browser* zur Darstellung der Benutzerschnittstelle.

Der Prozess der Adaption wird nachfolgend erläutert. Jeder der folgenden Abschnitte stellt dabei einen Teilprozess dar.

3.5.1. Die Transformation der multimodalen Benutzerschnittstelle

Startet eine EMODE Anwendung, so übergibt sie an den, durch die MSC erzeugten, Anwendungscontainer eine Referenz auf das eigene Arbeitsverzeichnis. Innerhalb des Arbeitsverzeichnisses sind alle Ressourcen der Anwendung abgelegt und stehen dem Anwendungscontainer zur Verfügung. Darunter befindet sich auch die multimodale Benutzerschnittstelle, deren Dialoge durch einzelne D3ML-Dokumente deklariert werden.

Nachdem eine Ausgabemodalität von der MSC erkannt und mit den Anforderungen der EMODE Anwendung verglichen wurde, wird sie dem Anwendungscontainer übergeben und steht der Anwendung zur Ausgabe von Informationen zur Verfügung. Sie wurde bisher jedoch noch nicht aktiviert. Die Ausgabedialoge müssen zuvor, durch den Anwendungscontainer, an die Ausgabemodalität übergeben werden, d. h. der Anwendungscontainer kontrolliert den Start- und Initialisierungsprozeß der Ausgabemodalität.

Die visuelle Ausgabe der multimodalen Benutzerschnittstelle, durch einen HTML-Browser, erfordert die Umwandlung der D3ML-Dialoge in eine äquivalente HTML-Darstellung. Die Umwandlung erfolgt durch sog. XSL-Transformationen mittels eines XSLT-Prozessors. Die Ausgabemodalität besitzt eine XSLT-Datei, darin sind Transformationsregeln enthalten, die angeben wie ein D3ML-Element in ein entsprechendes HTML-Element überführt wird. Die komplette Spezifikation wird in [Cla99] erläutert. Die Transformationen sind unabhängig vom eigentlichen Inhalt der Dokumente, dadurch kann die visuelle Ausgabemodalität für verschiedene Anwendungen verwendet werden. Die Umwandlung der D3ML-Dialoge wird innerhalb der Ausgabemodalität, in der Initialisierungsphase, durchgeführt.

Jedes Element der multimodalen Benutzerschnittstelle besitzt nach der XForms-Spezifikation, durch das `id`-Attribut, eine eindeutige Identifikation. Innerhalb der XSL-Transformation werden die Identifikationen der UI-Elemente übernommen, d. h. jedes Element der äquivalenten HTML-Darstellung besitzt die gleiche Identifikation wie das Gegenstück innerhalb der D3ML-Beschreibung. Allgemein gilt, dass das Ausgabeformat, welches durch die Ausgabemodalität mittels der XSL-Transformation erzeugt wird, ein Abbild der multimodalen Benutzerschnittstelle ist.

Der Grund für die Übernahme der Identifikationen der Elemente ist der, dass alle Modifikationen der multimodalen Benutzerschnittstelle, die durch die Benutzereingaben oder durch die Anwendungsausgaben auftreten, sich ebenfalls im *User Interface* der Ausgabemodalität widerspiegeln müssen. Werden neue Daten in das Element der multimodalen Benutzerschnittstelle mit der Identifikation „text1“ eingetragen, so geschieht dies ebenfalls im HTML-Dokument. Die visuelle Ausgabe bleibt somit immer in einem, zur multimodalen Benutzerschnittstelle, synchronen Zustand.

Kommt im Verlauf der Anwendung eine neue Ausgabemodalität hinzu, so bildet sie das eigene *User Interface* wiederum auf Basis der veränderten multimodalen Benutzerschnittstelle.

3.5.2. Die Modifikation der Benutzerschnittstelle im Format der Ausgabemodalität

Dieser Abschnitt widmet sich folgender Frage. Wie kann eine Ausgabemodalität beliebige Ausgaben der Anwendung im eigenen Format darstellen?

Zunächst soll geklärt werden, wie die Ausgabeinformationen von der Anwendung zur Ausgabemodalität gelangen. Das folgende Diagramm (Abb. 3.10) soll die einzelnen Prozesse der Verarbeitung der Anwendungsausgaben und deren Auswirkung auf die multimodale Benutzerschnittstelle

und den *User Interface*'s der Modalitäten darstellen. Die Dokumente der multimodalen Benutzerschnittstelle und die anwendungsspezifischen Transformationen (siehe 3.1.2.2) werden in der Startphase der EMODE Anwendung an den Applikationscontainer übergeben und sind im Diagramm als Voraussetzung zu beachten. Die einzelnen Schritte werden durch die Nummerierung gekennzeichnet und anschließend erläutert.

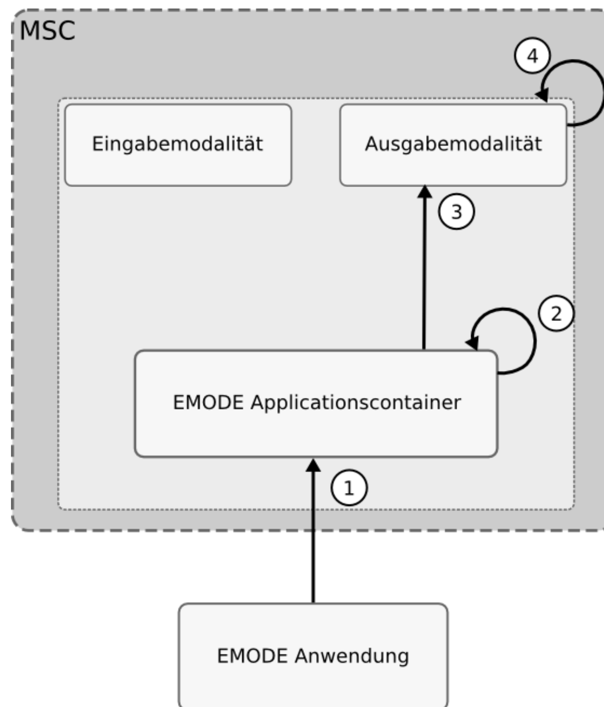


Abbildung 3.10.: Verarbeitung der Ausgabeinformationen einer EMODE Anwendung

(1) Jede EMODE Anwendung kann im Verlauf verschiedene Ausgaben erzeugen. Die Ausgabeanweisungen der Anwendung werden durch sog. *XUpdate*-Modifikationen beschrieben. Sie dienen dazu, neue Datenelemente in die Benutzerschnittstelle einzufügen, bestehende Datenelemente zu entfernen oder zu verändern.

Im Beispiel einer *Chat*-Anwendung wird eine empfangene Nachricht in den sog. *Message-Browser*-Bereich eingefügt. Er dient der Anzeige der versendeten und empfangenen Nachrichten. Die Anwendung muss den Inhalt der empfangenen Nachricht in eine *XUpdate*-Modifikationsbeschreibung so integrieren, dass der Anwendungscontainer weiß wie und wo die empfangene Nachricht, in die multimodale Benutzerschnittstelle, einzufügen ist (siehe dazu 3.2). Die Ausgabedaten der Anwendung werden durch die Mittel der Interprozesskommunikation an den Anwendungscontainer übergeben.

(2) Der Anwendungscontainer liest die empfangenen Ausgabeanweisung der EMODE Anwendung ein und übergibt die enthaltenen *XUpdate*-Modifikationen einer Verarbeitungskomponente, dem `D3MLDBManager`. Das Objekt ermittelt das betreffende D3ML-Dokument und führt die verschiedenen Operationen, die durch die *XUpdate*-Nachricht beschrieben werden, durch.

(3) Damit die Ausgabemodalitäten in einem äquivalenten Zustand bleiben, sendet der Anwendungscontainer die *XUpdate*-Modifikationen ebenfalls an alle zugeordneten Modalitäten. Das Listing 3.2 zeigt, dass die Ausgabeinformationen in einer allgemeinen XML-Beschreibung an die Ausgabemodalitäten gesendet werden. Die *XUpdate*-Modifikation kann jedoch in der Form nicht durch die Ausgabemodalität verwendet werden. Im Listing 3.2 soll ein neues *Client*-Element in eine Liste hinzugefügt werden. Damit das *Client*-Element

in eine HTML-Auswahllist eingefügt werden kann, muss es in eine äquivalente HTML-Darstellung umgewandelt werden. Das bedeutet, dass Datenelemente, die durch eine beliebige Ausgabemodalität dargestellt werden sollen, zunächst in das Ausgabeformat der Modalität transformiert werden müssen.

(4) Nachdem die Ausgabemodalität die empfangenen *XUpdate*-Modifikationen, mit Hilfe der anwendungsspezifischen Transformationsregeln, angepasst hat, werden sie an eine Verarbeitungskomponente übergeben. Sie führt anhand der *XUpdate*-Modifikationen Operationen in der Ausgabebeschreibung der aktuellen Modalität durch.

Zusammenfassend soll gelten, dass die *User Interface*'s der Ausgabemodalitäten parallel zur multimodalen Benutzerschnittstelle verändert werden. Die Abbildung 3.11 soll den Sachverhalt veranschaulichen.

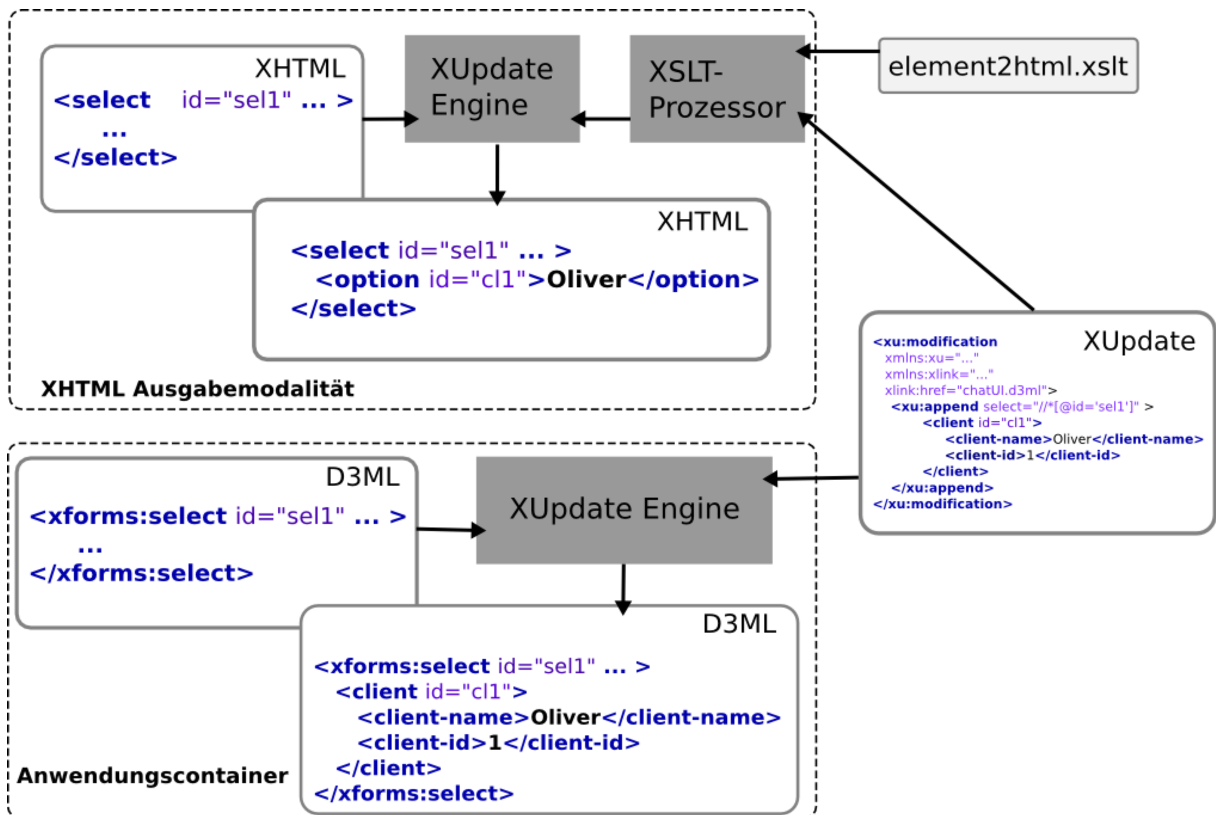


Abbildung 3.11.: parallele Modifikation der Benutzerschnittstellenbeschreibungen

3.6. Das dynamische Verhalten der MSC

In diesem Abschnitt werden die einzelnen Teilprozesse der MSC genauer beschrieben. Die folgende Abbildung A.4.1 zeigt, wie die einzelnen Systemakteure, bspw. die EMODE Anwendungen und die Modalitäten, zueinander in Beziehung stehen.

Eine EMODE Anwendung kann während des Ablaufs der MSC jederzeit gestartet und gestoppt werden. Die verschiedenen Anwendungen werden nebeneinander ausgeführt. Das gleiche gilt für die unterschiedlichen Ein- und Ausgabemodalitäten.

Die Laufzeitumgebung muss dafür sorgen, dass die nötigen Prozesse der Registrierung und Deregistrierung von Anwendungen und Modalitäten dynamisch durchgeführt werden.

In den folgenden Abschnitten werden die Teilprozesse der MSC genauer erläutert.

3.6.1. Aktivieren einer Modalität

Ein wichtiger Prozess der MSC ist das Einbinden von Modalitäten zur Laufzeit. Die Abbildung A.4.2 zeigt Teilprozesse, die während des Einbindens der Modalität stattfinden.

Jede Modalität wird im *OSGi-Framework* als *Bundle* aktiviert. Dabei soll das sog. „*White-board* Modell“ angewendet werden [Gat06]. Dies bedeutet, die Modalitäten müssen sich selbst während der Aktivierung, im *OSGi-Framework* registrieren. Der Vorteil dieser Vorgehensweise ist, dass die MSC nicht explizit Ausschau nach bestimmten Modalitäten halten muss, sondern nur dann reagiert, wenn bestimmte Modalitäten verfügbar werden. Dies vereinfacht den Mechanismus der automatischen Registrierung von Modalitäten innerhalb der MSC.

Durch das Registrieren der Modalität wird innerhalb des *Framework* ein sog. `ServiceEvent` ausgelöst. Die MSC kann durch das Objekt des `EMODEServiceListener` auf die verschiedenen Ereignisse (`ServiceEvent`) reagieren. Im Fall der Aktivierung einer Modalität wird ein bestimmtes Ereignis ausgelöst, welches durch das *OSGi-Framework* spezifiziert ist. Mit Hilfe des `ServiceEvent`-Objektes kann der `EMODEServiceListener` eine Referenz der Modalität erhalten und an die MSC übergeben, speziell an den `EMODEContainerManager`. Ist eine Modalität erfolgreich erkannt wurden, so wird sie im `EMODEContainerManager` als verfügbare Modalität angesehen.

Im nächsten Schritt überprüft der `EMODEContainerManager`, ob aktuell eine Anwendung existiert, die die verfügbar gewordene Modalität nutzen kann. Dazu wird ein Suchobjekt erzeugt, der `EMODEApplicationSearchThread`. Dieses Objekt erbt von der Java-Klasse `Thread` und ist dafür zuständig eine passende EMODE Anwendung zu finden. Dabei wird die Menge aller angemeldeten Applikationen untersucht. Das Objekt des `EMODEModalityApplicationConnector` ist dafür zuständig, die zu untersuchende EMODE Anwendung mit der bereitstehenden Modalität zu vergleichen und, wenn möglich, zu verbinden. Der Vergleich einer EMODE Applikation mit einer Modalität erfolgt durch das Bewerten der Ein- und Ausgabeeigenschaften der Modalität. Die EMODE Anwendung besitzt spezielle Anforderungen an die Modalität, die in Form von Eigenschaftsdaten an das Objekt des `EMODEModalityApplicationConnector` übergeben werden. Entsprechen die Eigenschaften der Modalität denen der Anwendung, so werden beide verknüpft und die Anwendung beginnt die Modalität zu nutzen. Der `EMODEContainerManager` vermerkt die Verbindung von Modalität zur Anwendung und entfernt die Modalität aus der Menge der verfügbaren Modalitäten, um so mehrfache Zugriffe zu verhindern.

Kann die aktuelle Modalität keiner EMODE Applikation zugeordnet werden, so verbleibt sie in der Menge der verfügbaren Modalitäten und steht somit weiteren Anwendungen zur Verfügung.

3.6.2. Deaktivieren einer Modalität

Ein ebenso wichtiger Prozess ist die Auskopplung und das Deaktivieren der Modalitäten. Die Abbildung A.4.3 stellt den Prozess und alle zugehörigen Komponenten schematisch dar.

Die Deaktivierung einer Modalität kann durch den Nutzer oder durch die MSC vorgenommen werden. Im letzteren Fall kann dies aufgrund einer automatischen Bewertung des aktuellen Kontext einer Anwendung geschehen. Als Beispiel dient die Ermittlung der aktuellen Umgebungslautstärke durch Umweltsensoren eines mobilen Endgerätes. Wird ein bestimmter Wert überschritten, so muss die Spracheingabemodalität deaktiviert werden. Der Prozess der automatischen Bewertung des Anwendungskontext wird in dieser Arbeit nicht weiter untersucht, da dies ein Schwerpunkt einer weiteren Forschungsarbeit ist.

Analog zur Aktivierung einer Modalität wird während der Deaktivierung ebenfalls ein `ServiceEvent`-Objekt innerhalb des *OSGi-Frameworks* ausgelöst. Der `EMODEServiceListener` der MSC reagiert auf das Ereignis der Deaktivierung und kann über das `ServiceEvent`-Objekt auf die Identifikation der jeweiligen Modalität zugreifen. Anhand der Identifikation der Modalität, kann das Objekt des `EMODEContainerManager` die aktuell zugeordnete EMODE Applikation ermitteln. Dies geschieht über eine Tabelle, in der jedem Anwendungscontainer die verwendeten Modalitäten zugeordnet werden.

Ist der entsprechende Anwendungscontainer ermittelt worden, so kann er und die betreffende Modalität an das Objekt des `EMODEModalityApplicationConnector` übergeben werden. Dieses ist nun dafür zuständig, die Nutzung der Modalität durch die EMODE Anwendung zu unterbrechen. Das Objekt des `EMODEApplicationContainer` verwaltet alle Modalitäten, die durch eine Anwendung genutzt werden. Der Transfer, der Ein- oder Ausgabedaten von der Anwendung zu einer Modalität, erfolgt innerhalb der `ModalityConnection`. Durch das Beenden und Auflösen der `ModalityConnection`, wird der Transfer der Ein- und Ausgabedaten gestoppt.

Nachdem die zu deaktivierende Modalität aus dem entsprechenden `EMODEApplicationContainer`-Objekt entfernt wurde, wird die Abbildung der Anwendung auf die Modalität ebenfalls aus der Zuordnungstabelles des `EMODEContainerManager` entfernt. Dies bedeutet, dass die aktuelle Modalität nun nicht mehr durch eine Anwendung genutzt wird.

Im letzten Schritt muss die zu deaktivierende Modalität aus der Menge der verfügbaren Modalitäten entfernt werden. Dies wird durch den `EMODEContainerManager` vorgenommen.

3.6.3. Integration der EMODE Anwendungen

Der Prozess der Integration der EMODE Anwendungen in die MSC wird durch die Abbildung A.4.4 verdeutlicht.

Verbindet sich eine EMODE Anwendung mit der MSC, so löst das Objekt des `EMODEContainerManager` ein Ereignis aus und delegiert die Behandlung dieses Ereignis an das Objekt des `EMODEApplicationListeners`. Dessen Aufgabe ist es, alle Schritte, die für die Integration der Anwendung in das System von Nöten sind, durchzuführen.

Im ersten Schritt der Integration der Anwendung werden alle nötigen Mittel für die Interprozesskommunikation bereitgestellt. Dazu gehört die Erzeugung des `EMODEEndpoint`, der für die Kommunikation der Anwendung mit der MSC zuständig ist. Des Weiteren wird für jede EMODE Anwendung eine Repräsentation innerhalb der MSC erzeugt, der `EMODEApplicationContainer`.

Nachdem alle Mittel für die Anwendung bereitgestellt worden, kann das Objekt des `EMODEApplicationContainer` beginnen, mögliche verwendbare Modalitäten zu suchen. Dafür wird das Objekt des `ModalitySearchThread` durch den `EMODEApplicationContainer` erzeugt. Es hat die Aufgabe, die Menge der verfügbaren Modalitäten mit den Anforderungen der EMODE Applikation zu untersuchen und die passende Modalität an die Applikation zu binden.

Zur Bindung einer passenden Modalität an die jeweilige Anwendung wird das Objekt des `EMODEModalityApplicationConnector`'s verwendet. Es vergleicht die Ein- und Ausgabeeigenschaften mit den Ein- und Ausgabeanforderungen der Anwendung und verbindet im positiven Fall die jeweilige Modalität mit der Anwendung (siehe dazu 3.6.1).

Der `EMODEContainerManager` vermerkt jede Bindung zwischen einer Modalität und einer Anwendung in der Zuordnungstabelle und entfernt die jeweilige Modalität aus der Menge der verfügbaren Modalitäten.

In dem Fall, dass keine passende Modalität für die Anwendung gefunden wird, so ist es dem Nutzer überlassen ob die Anwendung aktiv bleibt und somit im aktuellen Zustand in der MSC gehalten wird oder ob sie deaktiviert wird.

3.6.4. Beenden einer EMODE Anwendung

Die Abbildung A.4.5 zeigt die internen Vorgänge der MSC, während der Beendigung einer EMODE Anwendung.

Wenn eine EMODE Applikation beendet wird, so kann nur das Objekt des `EMODEEndPoint` dies erkennen. Es hat die Kontrolle über die Verbindung zur Anwendung und kann durch die Unterbrechung der Verbindung feststellen, wann eine Anwendung beendet wurde bzw. wann diese nicht mehr erreichbar ist und somit für die MSC als terminiert gilt. Daraufhin werden alle nötigen Schritte für die Beendigung der internen Prozesse, bezüglich der jeweiligen Anwendung, eingeleitet.

Der erste Schritt ist das Beenden der laufenden Modalitäten der jeweiligen Anwendung. Der `EMODEEndPoint` löst, bei Unterbrechung der Verbindung zur Anwendung, ein Ereignis aus. Dabei wird die zugehörige Instanz des `EMODEApplicationContainer`-Objektes übergeben.

Das Objekt des `EMODEApplicationListener`'s reagiert auf das Unterbrechungsereignis und weist den `EMODEModalityApplicationConnector` an, die Verbindungen zwischen der jeweiligen Anwendung und der durch sie genutzten Modalitäten zu unterbrechen. Dies bedeutet, dass der `EMODEApplicationContainer` alle ihm zugeordneten Modalitäten stoppen muss.

Nachdem alle Modalitäten gestoppt worden, hebt der `EMODEContainerManager` die Zuordnung der Modalitäten zur aktuellen Anwendung auf. Er entfernt alle betreffenden Einträge aus der internen Zuordnungstabelle und gibt die entsprechenden Modalitäten für die Nutzung durch andere Anwendungen frei.

Im zweiten Schritt wird der aktuelle Applikationscontainer zerstört und aus dem Register des `EMODEContainerManager` entfernt. Alle Kommunikationsressourcen werden dabei ebenfalls freigegeben.

4. Implementierung

In diesem Kapitel werden wichtige Funktionen und deren Implementierungen im Prototypen erläutert. Das Kapitel soll dabei nicht den gesamten Quellcode des Prototypen präsentieren, sondern einen Einblick in relevante funktionale Aspekte liefern.

Der erste Abschnitt des Kapitels soll die Vergleichsfunktion beschreiben, welche benötigt wird, um eine passende Modalität für eine EMODE Anwendung, aus der Menge aller verfügbaren Modalitäten, zu ermitteln. Danach folgt die Schilderung des Verbindungsaufbaus zwischen einer Modalität und einer EMODE Anwendung. Im darauf folgenden Abschnitt wird die Analyse der Eingabedaten, innerhalb des Anwendungscontainers, erklärt. Der letzte Abschnitt präsentiert den Mechanismus zum Aufruf der Anwendungsfunktionen anhand von Kommandos, innerhalb einer EMODE Beispielanwendung, die in der Skriptsprache Ruby implementiert ist. Das Ziel ist, zu zeigen, dass der Aufruf der Funktionen nach dem Entwurf der 2. Variante im Abschnitt 3.3 anwendbar ist.

4.1. Der Vergleich der Anwendungsanforderung mit den Eigenschaften der Modalität

In den Abschnitten 3.6.1und 3.6.3 werden die Vorgänge der Modalitäten- und Anwendungsregistrierung beschrieben. Beide Vorgänge besitzen das Ziel eine Verknüpfung, zwischen einer EMODE Anwendung und einer Modalität, herzustellen. In beiden Prozessen ist dabei das Objekt des `EMODEModalityApplicationConnector`'s beteiligt. Es vergleicht die Eigenschaften der Modalität mit den Anforderungen der EMODE Anwendung. Das Ziel des Vergleichs ist die Ermittlung einer geeigneten Modalität aus der Menge aller verfügbaren Modalitäten.

Wenn eine Modalität durch die MSC erkannt und registriert wurde (siehe 3.6.1), so ist das Objekt des `EMODEApplicationSearchThread`'s dafür zuständig, nach einer passenden registrierten EMODE Anwendung zu suchen. Dazu iteriert es über die Menge aller erzeugten `EMODEApplicationContainer`-Objekte (Listing 4.1 Zeile 4-16). Jedes `EMODEApplicationContainer`-Objekt ist eine Repräsentation der zugehörigen EMODE Anwendung innerhalb der MSC.

Startet eine EMODE Anwendung, so wird sie durch die MSC registriert und es wird ein zugehöriges `EMODEApplicationContainer`-Objekt erzeugt. Nachdem alle Ressourcen in den `EMODEApplicationContainer` geladen wurden, kann die Suche nach geeigneten Modalitäten beginnen. Dazu wird der `ModalitySearchThread` durch den Anwendungscontainer erzeugt. Er sucht, in der Menge aller registrierten Modalitäten, nach geeigneten Einheiten (Listing 4.2). Wenn eine Modalität für gültig befunden und mit dem `EMODEApplicationContainer`-Objekt verbunden wurde, wird sie aus der Menge der verfügbaren Modalitäten entfernt (Listing 4.1 Zeile 12 und Listing 4.2 Zeile 11).

Der `EMODEModalityApplicationConnector` kann über die Methode `getEMODEModalityApplicationConnector()` des `EMODEContainerManager` erhalten werden (Listing 4.1 Zeile 2-3 und Listing 4.2 Zeile 2-3).

Der Vergleich der aktuell frei verfügbaren Modalität mit den Anforderungen der gegenwärtigen EMODE Anwendung wird durch den Aufruf der Methode `compare(...)` des `EMODEModality-ApplicationConnector`'s gestartet (Listing 4.1 Zeile 7). Der boolsche Rückgabewert der Vergleichsmethode zeigt die Eignung der Modalität, in Bezug auf die Anforderungen der EMODE Anwendung, an. Die Implementierung des Vergleichsalgorithmus wird im folgenden Abschnitt 4.1.1 anhand des Quellcodes erläutert.

Listing 4.1: `EMODEApplicationSearchThread run()`-Methode

```
1  public void run() {
2    EMODEModalityApplicationConnector connector = manager
3                                .getEMODEModalityApplicationConnector();
4    Iterator<EMODEApplicationContainer> app_iter = applications
5                                .iterator();
6    while(app_iter.hasNext()){
7      EMODEApplicationContainer cur_container = app_iter.next();
8      if(connector.compare(modality, cur_container)){
9        // modality meets conditions of application
10       synchronized (connector) {
11         try {
12           // connect modality with current application container
13           connector.connect(modality, cur_container);
14           manager.getAllModalitiesNotInUse().remove(modality);
15         } catch (Exception e) { e.printStackTrace(); }
16       }
17       break; // stop iteration
18     }
19   }
20 }
```

Listing 4.2: `ModalitySearchThread run()`-Methode

```
1  public void run() {
2    EMODEModalityApplicationConnector connector = manager
3                                  .getEMODEModalityApplicationConnector();
4    Iterator<IModality> iter = modalities.iterator();
5    while(iter.hasNext()){
6      IModality cur_modality = iter.next();
7      if(connector.compare(cur_modality, application)){
8        synchronized (connector) {
9          try {
10           connector.connect(cur_modality, application);
11           // remove modality from set of available modalities
12           iter.remove();
13         } catch (Exception e) {e.printStackTrace();}
14       }
15     }
16   }
17 }
```

4.1.1. Der Vergleichsalgorithmus

Der Vergleichsalgorithmus ist von entscheidender Bedeutung, denn er bestimmt die Gültigkeit der jeweiligen Modalität in Bezug auf die Anforderungen der EMODE Anwendung und hat somit einen entscheidenden Einfluss auf die Verfügbarkeit der Modalität gegenüber der EMODE Anwendung. Dementsprechend soll er in diesem Abschnitt anhand des Quellcodes erläutert werden.

Der Vergleich der Eigenschaften der aktuellen Modalität mit den Anforderungen der EMODE Anwendung erfolgt über die Daten, die in den Meta-Informationen des jeweiligen Vergleichsobjekt enthalten sind. Durch die Methoden `wantedInputModalityProperties()` und `wantedOutput-ModalityProperties()` können die Anforderungen der EMODE Anwendung, bezüglich der Ein- und Ausgabeeigenschaften der zu verwendenden Modalitäten, ermittelt werden (Listing B.1 Zeile 4-6). Die Anforderungen der EMODE Anwendung werden mit Hilfe einer Tabelle definiert. Die Tabelle enthält als Einträge Schlüssel/Wert-Paare. Der Schlüssel eines Eintrags ist der Name eines Attributs, welches die Modalität ebenfalls besitzen muss. Der Wert des Eintrags ist eine Liste von möglichen Werten, die das Attribut bzw. die Modalität besitzen kann.

Die Ein- und Ausgabeeigenschaften der aktuellen Modalität werden durch den Aufruf der Methoden `getInputProperties()` und `getOutputProperties()` (Listing B.1 Zeile 18-21) bereitgestellt.

Im Abschnitt (3.2.1) werden die Modalitäten in 3 verschiedene Kategorien unterteilt. Je nach Kategorie unterscheiden sich die Ein- und Ausgabeeigenschaften der Modalität. Eine Eingabemodalität besitzt keine Ausgabefunktionalität, dementsprechend ist die Menge der Ausgabeeigenschaften die leere Menge.

Die Kategorie der aktuellen Modalität kann anhand des Zustands bestimmter Objekte erkannt werden (Listing B.1 Zeile 26; 33; 40). Ist das Objekt, welches für die Interpretation der Eingabedaten zuständig ist (`EMMAProcessor`), mit dem Wert `null` belegt, so ist die zu untersuchende Modalität keine Eingabemodalität und somit entweder eine Ausgabemodalität oder ungültig. Ist in diesem Fall das Objekt, welches den Zugriff auf die Ausgabedaten durchführt (`UIAccessor`), ungleich dem Wert `null`, so kann es sich bei der aktuellen Modalität nur um eine Ausgabemodalität handeln. Tritt der Fall auf, dass der `EMMAProcessor` und der `UIAccessor` ungleich `null` sind, so handelt es sich bei der Modalität um eine Ein-/Ausgabemodalität (siehe Abschnitt 3.2.1).

Nachdem die Kategorie der aktuellen Modalität bestimmt wurde, kann die Untersuchung der Eigenschaften der aktuellen Modalität beginnen. Die Methode `checkAttributeKeys(...)` (Listing B.1 Zeile 80 - 108) hat die Aufgabe herauszufinden, ob die Attribute der aktuellen Modalität und deren Werte in der Menge aller möglichen Attribute enthalten sind. Dazu wird eine *Checkliste* verwendet, die jedem Attributsnamen der Modalität einen Wahrheitswert zuordnet (Listing B.1 Zeile 23).

Zunächst muss das Vorkommen des Attributs, aus der Eigenschaftstabelle der aktuellen Modalität, in der Anforderungstabelle der EMODE Anwendung bestätigt werden (Listing B.1 Zeile 94). Ist dies der Fall, so muss der Wert des Attributs, aus der Eigenschaftstabelle der aktuellen Modalität, untersucht werden. Die Liste aller möglichen Werte eines Attributs ist, in der Anforderungstabelle, dem Attributsnamen zugeordnet. Ist der Wert des Attributs, aus der Eigenschaftstabelle der aktuellen Modalität, in der Liste aller möglichen Werte enthalten, so kann das Attribut als wahr markiert werden. Diese Funktion wird durch die Methode `markKey(...)` implementiert (Listing B.1 Zeile 110-129). Als Argumente erhält die Methode die Liste aller erlaubten Attributswerte, den Wert des aktuellen Attributs (der zu vergleichenden Modalität) und den zu markierenden Attributnamen (Schlüssel). Als Ergebnis wird ein Eintrag, bestehend aus dem Attributnamen und einem zugehörigen boolschen Wert, der die Gültigkeit des Attributs anzeigt, zurückgegeben (Listing B.1 Zeile 103). Der Attributwert (der zu untersuchenden Modalität) kann ein `String` oder eine Liste von `String`'s sein. Die Ursache dafür zeigt die Tabelle 4.1 der Eingabeeigenschaften einer Ein-/Ausgabemodalität.

Das Attribut `device` (Tabelle 4.1) besitzt als Wert eine Liste von Eingabegeräten, die durch die Modalität repräsentiert werden. Das Attribut `medium` hingegen besitzt als Wert einen `String`, der lediglich das Eingabemedium benennt.

Attributname	Attributwerte
device	[mouse, keyboard]
medium	tactile
...	...

Tabelle 4.1.: Eingabeeigenschaften einer Ein- / Ausgabemodalität

Die Unterscheidung der möglichen Datentypen des Attributwertes wird in der Methode `markKey(...)` vorgenommen (Listing B.1 Zeile 116-123). Im darauf folgenden Schritt wird überprüft, ob der Wert des aktuellen Attributs in der Liste aller erlaubten Werte enthalten ist. Im positiven Fall wird das Attribut in einem Eintrag als wahr markiert. Im negativen Fall wird es im Eintrag als falsch markiert und anschließend zurückgegeben (Listing B.1 Zeile 126-128). Der Eintrag, bestehend aus dem markierten Attributnamen, wird nun in die *Checkliste* eingefügt (Listing B.1 Zeile 104). Dies geschieht für jedes Attribut der entsprechenden Eigenschaftstabelle der zu untersuchenden Modalität. Im Fall einer Ein-/Ausgabemodalität müssen beide Eigenschaftstabellen, die der Eingabe- und der Ausgabefunktionalität, untersucht werden.

Nachdem für jedes Attribut der Modalität ein Eintrag mit einem boolschen Wert erzeugt und in die *Checkliste* eingefügt wurde, kann diese nun auf die Allgemeingültigkeit überprüft werden. Damit die zu untersuchende Modalität als gültig anerkannt wird, müssen alle Attribute in der *Checkliste* als wahr markiert sein. Die Untersuchung der Attribute erfolgt durch die Iteration über die *Checkliste*. Wird ein Eintrag gefunden der nicht als wahr markiert ist, so wird die Modalität als nicht gültig anerkannt (Listing B.1 Zeile 61-70). Sind alle Einträge der *Checkliste* wahr, so wird die Modalität als gültig anerkannt (Listing B.1 Zeile 75).

Im nächsten Schritt wird die gültige Modalität mit dem ausgewählten `EMODEApplicationContainer`-Objekt durch die `connect(...)`-Methode des `EMODEModalityApplicationConnector`'s verbunden (Listing 4.1 Zeile 13).

Es kann der Spezialfall auftreten, dass identische Modalitäten zur selben Zeit innerhalb MSC registriert sind. Dies kann passieren, wenn parallel ablaufende EMODE Anwendungen die selben Modalitäten benötigen. Damit die Modalitäten gerecht an die Applikationen verteilt werden, muss der `EMODEModalityApplicationConnector` doppelte Zuordnungen verhindern. Dies geschieht, indem er während des Vergleichs die Attributwerte aus der Anforderungstabelle entfernt (Listing B.1 Zeile 118; 122). In einem weiteren Vergleich mit einer identischen Modalität sind die Attributwerte nicht mehr in der Anforderungstabelle enthalten. Ein Attribut der identischen Modalität wird demnach als falsch markiert. Die Modalität wird am Ende des Vergleichs als ungültig erkannt und steht somit weiteren Anwendungen zur Verfügung.

Durch die Iteration werden die Attribute der aktuellen Modalität sukzessiv untersucht. Wenn der Wert des Modalitätenattributs dem Wert des Attributs in der Anforderungstabelle entspricht, so wird es aus der Anforderungstabelle entfernt. Dies geschieht auch dann wenn die Modalität die gesamten Anforderungen nicht erfüllt, also ungültig ist. Der folgende Spezialfall soll dies verdeutlichen.

Es wird eine Modalität untersucht, deren Attributwerte, außer der Letzte, denen in der Anforderungstabelle entsprechen. Durch die Iteration werden fast alle Werte aus der Anforderungstabelle entfernt. Da das letzte Attribut nicht den Anforderungen entspricht, muss die Modalität als ungültig erkannt werden, d. h. die entfernten Werte der Anforderungstabelle müssen wieder hergestellt werden. Dies geschieht indem die Werte mit Hilfe der zuvor gesicherten Kopien (Listening B.1 9-16) in die Anforderungstabelle eingefügt werden. Diese Funktion wird durch die `rollback(...)`-Methode implementiert (Listening B.1 Zeile 66-67).

Der Verbindungsaufbau zwischen einer Modalität und einer EMODE Anwendung wird im nächsten Abschnitt erläutert.

4.2. Der Verbindungsaufbau zwischen Modalität und EMODE Anwendung

Wenn eine EMODE Anwendung mit einer Modalität verknüpft werden soll, so muss zunächst, wie im vorherigen Abschnitt beschrieben, die Gültigkeit bzw. die Ungültigkeit der Modalität gegenüber den Anforderungen der EMODE Anwendung bestätigt werden.

Nachdem eine Modalität gegenüber den Anforderungen der EMODE Anwendung für gültig befunden wurde, wird sie durch die `connect(...)`-Methode des `EMODEModalityApplicationConnector`'s mit dem Objekt des `EMODEApplicationContainer`'s, verbunden. Dabei wird die Methode `startUsingService(...)` aufgerufen und die Schnittstelle der Modalität, vom Typ `IModality`, als Argument übergeben (Listing 4.3).

Jede zu nutzende Modalität wird im Anwendungscontainer in eine Liste eingetragen (Listing 4.3 Zeile 2). Sie enthält alle dem Anwendungscontainer zugeordneten Modalitäten und dient der späteren Deaktivierung einzelner bzw. aller Modalitäten.

Nach Abschnitt 3.5.1 muss der Anwendungscontainer die Beschreibung der multimodalen Benutzerschnittstelle, die D3ML-Dokumente, und die anwendungsspezifischen Transformationen (siehe 3.3.3.1 und 3.1.2.2) vor der Aktivierung der Modalität übergeben, d. h. er muss die Kontrolle über die Initialisierungs- und Startphase der Modalität besitzen (Listing 4.3 Zeile 7-15).

Nachdem der Anwendungscontainer alle notwendigen Daten an die Modalität übergeben und aktiviert hat, erzeugt er ein `ModalityConnection`-Objekt, welches die Bereitstellung der Kommunikationsmittel und die Abläufe der Kommunikation in sich kapselt (Listing 4.3 Zeile 20).

Die Verbindung zwischen `EMODEApplicationContainer` und einer Modalität hängt von der Kategorie der Modalität ab. Handelt es sich bei der Modalität um eine Eingabemodalität, so kann der `EMODEApplicationContainer`, innerhalb der Verbindung, ausschließlich Daten empfangen. Ähnlich verhält es sich im Fall einer Ausgabemodalität, der Anwendungscontainer kann innerhalb der Verbindung ausschließlich senden. Kann die Modalität der Kategorie der Ein-/Ausgabemodalitäten zugeordnet werden, so besitzt die Verbindung zwischen dem `EMODEApplicationContainer` und der Modalität einen dualen Charakter, d. h. sie kann simultan Daten Senden und Empfangen.

Listing 4.3: `EMODEApplicationContainerImpl startUsingService(...)`-Methode

```
1  public void startUsingService(IModality imodality) throws Exception {
2    this.modalities_inuse.add(imodality);
3
4    ...
5
6    // initialize modality with data
7    imodality.setUIDescription(getD3MLDescriptions());
8    imodality.setApplicationLocation(this.app_location);
9    imodality.init();
10
11   // give contextual transformations to the modality
12   giveContextualDataToModality(imodality);
13
14   // start modality
15   imodality.run();
```

```
16
17    ...
18
19    // create new connection between this container and modality
20    ModalityConnection m_connector = new ModalityConnection(
21                                  (EMODEApplicationContainer)this,
22                                  imodality,
23                                  application);
24
25    this.connections.add(m_connector);
26  }
```

4.2.1. Das ModalityConnection-Objekt

Die Vorgänge des Senden und Empfangen werden im `ModalityConnection`-Objekt durch die privaten Objekte `ModalityOutput` (Senden) und `ModalityInput` (Empfangen) repräsentiert (Listing 4.4 Zeile 11; 19). Beide Objekte erben von der Java-Klasse `java.lang.Thread` und ermöglichen somit einen parallelen Ablauf.

Listing 4.4: `ModalityConnection`-Konstruktor

```
1  public ModalityConnection(EMODEApplicationContainer parent,
2                            IModality modality,
3                            EMODEEndPoint endpoint){
4
5    this.parent = parent;
6    this.service = modality;
7    this.endpoint = endpoint;
8
9    // input reading
10   if(service.getEMMAProcessor() != null){
11     this.input = new ModalityInput(
12                           service.getEMMAProcessor().openConnection());
13     this.input.start();
14   }
15
16   // output writing
17   if(service.getUIAccessor() != null){
18     this.outputmessages = new LinkedList<String>();
19     this.output = new ModalityOutput(
20                           service.getUIAccessor().openConnection(), this);
21     this.output.start();
22   }
23 }
```

ModalityInput

Damit die Eingabedaten einer Modalität empfangen werden können, wird eine Art Verbindungskanal zum `EMMAProcessor` der Eingabemodalität geöffnet. Nach Abschnitt 3.4.1 erzeugt der `EMMAProcessor` die Eingabeinterpretation und stellt sie dem Anwendungscontainer bereit. Der Verbindungskanal besteht auf der einen Seite aus einem `PipedOutputStream` und auf der anderen Seite aus einem `PipedInputStream`. Der `EMMAProcessor` schreibt auf der Seite der Modalität alle interpretierten Daten in den `PipedOutputStream`. Auf der Seite des Anwendungscontainers liest der `ModalityInput-Thread`, innerhalb des `ModalityConnection`-Objektes, alle Eingabedaten, mit Hilfe des durch die `openConnection()`-Methode übergebenen `PipedInputStream`, ein

(Listing 4.4 Zeile 12). Die interpretierten Eingabedaten werden innerhalb der `run()`-Methode des `ModalityInput`-Objektes zur weiteren Untersuchung an das `EMODEInputMessageProcessor`-Objekt übergeben (Listing B.2 Zeile 16). Die Implementierung der `processMessage(...)`-Methode wird im Abschnitt 4.3 noch erläutert.

Nachdem die Daten durch das Objekt verarbeitet worden sind, werden sie an die EMODE Anwendung durch den `EMODEEndPoint` weitergeleitet (Listing B.2 Zeile 22).

ModalityOutput

Damit die Ausgabeinformationen des Anwendungscontainer versendet werden können, wird ein Verbindungskanal zum `UIAccessor` der aktuellen Ausgabemodalität geöffnet. Das Objekt des `UIAccessor`'s ist dafür zuständig neue Informationen bzw. Elemente in das *User Interface* der Modalität zu integrieren, bestehende Informationen bzw. Elemente zu verändern oder zu entfernen. Die *XUpdate*-Modifikationen werden vom Anwendungscontainer über den Verbindungskanal an den `UIAccessor` übergeben. Der Verbindungskanal besteht auf der Seite der Ausgabemodalität aus einem `PipedInputStream` und auf der Seite des Anwendungscontainers aus einem `PipedOutputStream`.

Der Anwendungscontainer bezieht die Ausgabeinformationen aus 2 möglichen Quellen. Die erste Nachrichtenquelle ist die zugehörige EMODE Anwendung. Die Ausgabeinformationen der EMODE Anwendungen können mit Hilfe des `EMODEEndPoint` erhalten werden (Listing B.3 Zeile 6). Die zweite Quelle ist ein Puffer, der Nachrichten anderer Modalitäten bzw. des `EMODE-InputMessageProcessor`-Objektes, enthalten kann (Listing B.3 Zeile 9). Als Beispiel dient die Spracheingabemodalität, deren Eingabe visuell durch eine Ausgabemodalität dargestellt werden soll. In diesem Fall soll die Darstellung der Eingabe dem Nutzer als *Feedback* dienen. Die Nachrichten anderer Ausgabemodalitäten müssen durch den Puffer aufgefangen werden, der Grund dafür ist folgender.

Dem Anwendungscontainer können gleichzeitig mehrere Ausgabemodalitäten zugeordnet sein. Jede Modalität besitzt durch den `EMODEEndPoint` Zugriff auf die Ausgabenachrichten der Anwendung. Durch die Synchronisation des Zugriffs, erhält nur eine Ausgabemodalität die Nachrichten, welche der `EMODEEndPoint` bereitstellt. Damit jede Ausgabemodalität die Ausgabeinformationen erhalten kann, muss die Modaltität, die die Informationen vom `EMODEEndPoint` erhalten hat, ein *Multicast* an alle anderen Ausgabemodalitäten durchführen (Listing B.3 Zeile 21). Diese Aufgabe wird mit Hilfe der Liste aller Modalitäten des `EMODEApplicationContainer`-Objektes erfüllt.

Jede Ausgabenachricht ist eine Beschreibung der Modifikation der multimodalen Benutzerschnittstelle im *XUpdate*-Format. Durch die Methode `processXUpdateMessages(...)` (Listing B.3 Zeile 15) werden die *XUpdate*-Anweisungen an das Objekt übergeben, welches für die Ausführungen der Modifikationen im jeweiligen D3ML-Dokument zuständig ist (`D3MLDBManager`).

Die D3ML-Dokumente beinhalten das Datenmodell der EMODE Anwendung. Sollen neue Informationselemente in die Ausgabe hinzugefügt oder sollen bestehende Elemente verändert bzw. entfernt werden, so wird dies in der multimodalen Benutzerschnittstelle ausschließlich über die Elemente des Datenmodells durchgeführt. In der konkreten Darstellung der Ausgabeinformationen, z.B. durch ein HTML-Dokument, werden die Modifikationen direkt über die Elemente des *User Interface* vollzogen. Damit ergibt sich ein Problem, denn die *XUpdate*-Modifikationsanweisungen beziehen sich ausschließlich auf die D3ML-Dokumente und nicht auf die Ausgabeformate der Ausgabemodalitäten. Im Listing 3.2 in Abschnitt 3.1.2.2 ist eine *XUpdate*-Modifikation dargestellt, die ein neues Informationselement in das Datenelement mit der Identifikation `m_clientlist` eintragen soll. Damit die *XUpdate*-Anweisung ebenfalls an die Ausgabemodalitäten verteilt werden

kann, muss das zu verändernde Element ein UI-Element der Ausgabemodalität sein. Jede Ausgabemodalität übernimmt nach Abschnitt 3.5.1 die Identifikation jedes UI-Elementes in die eigene Darstellung der Ausgabe.

Damit nun die *XUpdate*-Modifikationen auch auf die Ausgabe der Modalitäten angewendet werden können, müssen die angezeigten Datenmodellelemente (das `select`-Attribut) durch die UI-Elemente ersetzt werden, d. h. die Bindung zwischen Datenmodellelement und UI-Element wird aufgelöst. Dies geschieht in der Methode `processXUpdateMessages(...)`. Der Rückgabewert der Methode ist die veränderte XUpdate-Anweisung (Listing B.3 Zeile 15).

Im nächsten Schritt werden die Ausgabenachricchten an die Modalitäten versendet (Listing B.3 Zeile 24; 30). Dies geschieht indem die `println(...)` Methode des `PipedOutputStream`'s des `UIAccessor`'s der Ausgabemodalität aufgerufen wird.

Nachdem nun die Kommunikationsverbindungen zwischen der Modalität und dem Anwendungscontainer bereitstehen, können die Eingabe- und Ausgabeinformationen ausgetauscht werden.

Der nächste Abschnitt widmet sich speziell der Verarbeitung der Eingabedaten, durch den `EMODE-ApplicationContainer`.

4.3. Die Analyse der Eingabedaten durch den Anwendungscontainer

Der Nachrichtenstrom der Eingabemodalitäten kann nach Abschnitt 3.1.2.1 verschiedene *Markup*-Aktionen enthalten. Eine Übersicht ist in der Tabelle 3.2 in Abschnitt 3.4.2 dargestellt.

Der `EMODEApplicationContainer` hat die Aufgabe, die *Markup*-Elemente aus dem Strom der Eingabenachrichten zu ermitteln, weiter zu verarbeiten und gegebenfalls mögliche Ergebnisse der Verarbeitung in den Datenstrom der Eingabe zurückzuführen. Für diese Aufgabe besitzt der Anwendungscontainer das Objekt `EMODEInputMessageProcessor`. In Zeile 16 des Listing B.2 wird die Nachricht der Eingabemodalität an den `EMODEInputMessageProcessor` übergeben. Dies geschieht durch den Aufruf der Methode `processMessage(...)`. Durch sie werden die verschiedenen *Markup*-Aktionen aus der Eingabenachricht ermittelt und weitere Verarbeitungsfunktionen aufgerufen. Die Abbildung 4.1 verdeutlicht die Verzweigung der internen Verarbeitungsschritte und dient dem besseren Verständnis des Funktionsablaufs. Die Methode `processMessage(...)` ist der Ausgangspunkt für die weitere Verarbeitung der verschiedenen Teilelemente der Eingabenachrichten.

Die Aktionsbeschreibungen, die in der Eingabenachrichten vorkommen können, lassen sich in folgende zwei Kategorien aufteilen.

XForms-Aktionen dienen der Manipulation der multimodalen Benutzerschnittstelle.

EMODE-Aktionen dienen dem Abfragen von Elementen der multimodalen Benutzerschnittstelle und dem Öffnen neuer UI-Dialoge.

Die Abbildungen unter den nachfolgenden Abschnitten verdeutlichen die Implementationen der Verarbeitungsfunktionen. Die einzelnen Nummerierungen in den Abbildungen dienen dazu, die jeweiligen Teilprozesse genauer zu erklären.

Die verschiedenen Aktionselemente der Eingabenachricht (siehe Abb. 4.1) werden mit Hilfe der *XPath*-Technologie ermittelt (Abb. A.5.1 Punkt 1.1 und 1.2). Die Eingabenachricht wird zuvor in ein Datenstrukturbaum überführt (siehe Abb. A.5.1 Punkt 1). In dieser Arbeit wird dazu JDOM verwendet. Es ist eine Java optimierte API, zum Lesen und Schreiben von XML Daten.

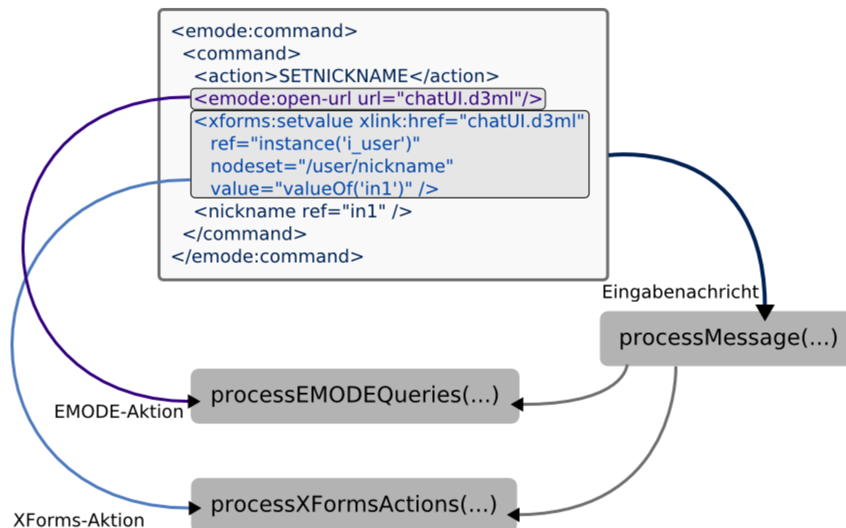


Abbildung 4.1.: Verarbeitung der Eingabenachrichten

Die *XPath*-Ausdrücke erzeugen, je nach Aktionskategorie, eine Liste von Aktionselementen, die in der Eingabenachricht vorkommen. Zur Verarbeitung der Liste der EMODE-Aktionen wird die Methode `processEMODEQueries(...)` aufgerufen (Abb. 4.1 und A.5.1 Punkt 2.2). Die Liste der XForms-Aktionen wird durch die Methode `processXFormsActions(...)` verarbeitet (Abb. 4.1 und A.5.1 Punkt 2.1).

4.3.1. Die Verarbeitung der EMODE-Aktionen

Die Menge der EMODE-Aktionen besteht nach der Tabelle 3.2 Abschnitt 3.4.2 aus den Elementen `<emode:value-of>` und `<emode:open-url>`. Die Beschreibung der Elemente ist ebenfalls in der Tabelle 3.2 enthalten.

Die Verarbeitung des `<emode:value-of>` Elementes erfolgt durch die Methode `processEMODE-ValueOfQuery(...)` (Abb. 4.2 Punkt 2.2). Sie soll durch den folgenden Abschnitt erläutert werden. Die Methode `processEMODEOpenURLQuery(...)` dient dem Auswerten der `<emode:open-url>` Aktionen (Abb. 4.2 Punkt 2.1).

Die Verarbeitung der `<emode:value-of>`-Nachricht

Die Methode `processEMODEValueOfQuery(...)` erhält als Argumente die Aktionsbeschreibung vom Datentyp `Element` (JDOM-API) und eine Tabelle, die das Ergebnis der Abfrage dem Aktionselement zuordnet. Der Grund dafür wird noch erläutert.

Zunächst wird das Attribut bestimmt, welches den Namen des betreffenden D3ML-Dokumentes, der multimodalen Benutzerschnittstelle, enthält (`xlink:href`-Attribut des `<emode:value-of>` Elementes). Damit wird das D3ML-Dokument, für die Abfrage von Elementen, bereitgestellt (Abb. 4.3 Punkt 2). Des Weiteren muss das abzufragende Element bestimmt werden. Es wird durch das `select`-Attribut der `<emode:value-of>`-Beschreibung angezeigt (Abb. 4.3 Punkt 1). Das Attribut enthält ein *XPath*-Ausdruck, der dazu dient den Wert des angezeigten Elements, im JDOM-Baum (`select`-Attribut) des D3ML-Dokuments, zu ermitteln. (Abb. 4.3 Punkt 3).

Nachdem das Abfrageergebnis bereitsteht, kann nun die Tabelle, welche als Argument übergeben wurde, zum Einsatz kommen. Sie ist notwendig, damit das Resultat der Abfrage in den Strom der Eingabedaten zurückgeführt werden kann. Nachdem jeder `<emode:value-of>`-Abfrage ein

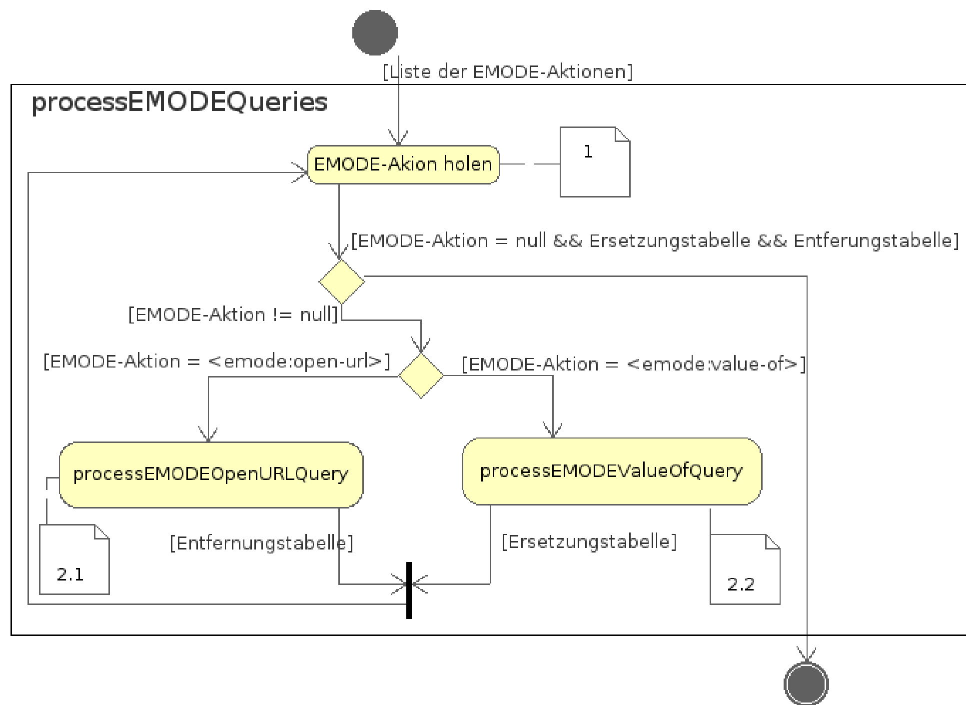


Abbildung 4.2.: processEMODEQueries(...) Methode

Ergebnis zugeordnet wurde, wird die Tabelle dazu verwendet, um die Abfrageelemente in der Eingabenachricht durch die zugeordneten Resultate zu ersetzen (Abb. 4.3 Punkt 4). Ist die Verarbeitung beendet, so enthält die Eingabenachricht alle Ergebnisse der Abfrageelemente und kann im nächsten Schritt an die Anwendung übergeben werden. Die Ersetzung der `<emode:value-of>`-Abfragen wird durch die Abbildung A.5.2 veranschaulicht.

Die Verarbeitung der `<emode:open-url>`-Nachricht

Wenn in der Eingabenachricht das Element `<emode:open-url>` auftaucht, so bedeutet dies, dass der gegenwertige UI-Dialog durch einen neuen ersetzt werden muss. Das `url`-Attribut des `<emode:open-url>`-Elementes zeigt dabei auf das zu verwendende D3ML-Dokument (siehe Abb. 4.1).

Im Verarbeitungsprozess der Nachricht geht es hauptsächlich um die Bestimmung der neuen Dialog-Datei und den Wechsel der UI-Dialoge in den Ausgabemodalitäten.

Der neue Dialog wird mit Hilfe des `url`-Attributwertes bestimmt (Abb. 4.4 Punkt 1). Er kann durch den Aufruf der Methode `setAsCurrentStartUI(...)` als aktueller Dialog gekennzeichnet werden, dabei wird das zugehörige D3ML-Dokument markiert (Abb. 4.4 Punkt 3). Jede Modalität die ab diesem Zeitpunkt in den Verlauf des Anwendungscontainers hinzugefügt wird, erkennt anhand der Markierung, welches D3ML-Dokument den aktuellen Start-Dialog enthält.

Alle Ausgabemodalitäten, die bereits durch den Anwendungscontainer benutzt werden, erhalten den neuen UI-Dialog durch den Aufruf der `UIAccessor`-Methode `setCurrentDialog(...)` (Abb. 4.4 Punkt 4), sie führen anschließend den Wechsel der UI-Dialoge selbstständig durch.

Der Wechsel der UI-Dialoge findet auf der Ebene der multimodalen Benutzerschnittstelle und auf der Ebene der Modalitäten statt.

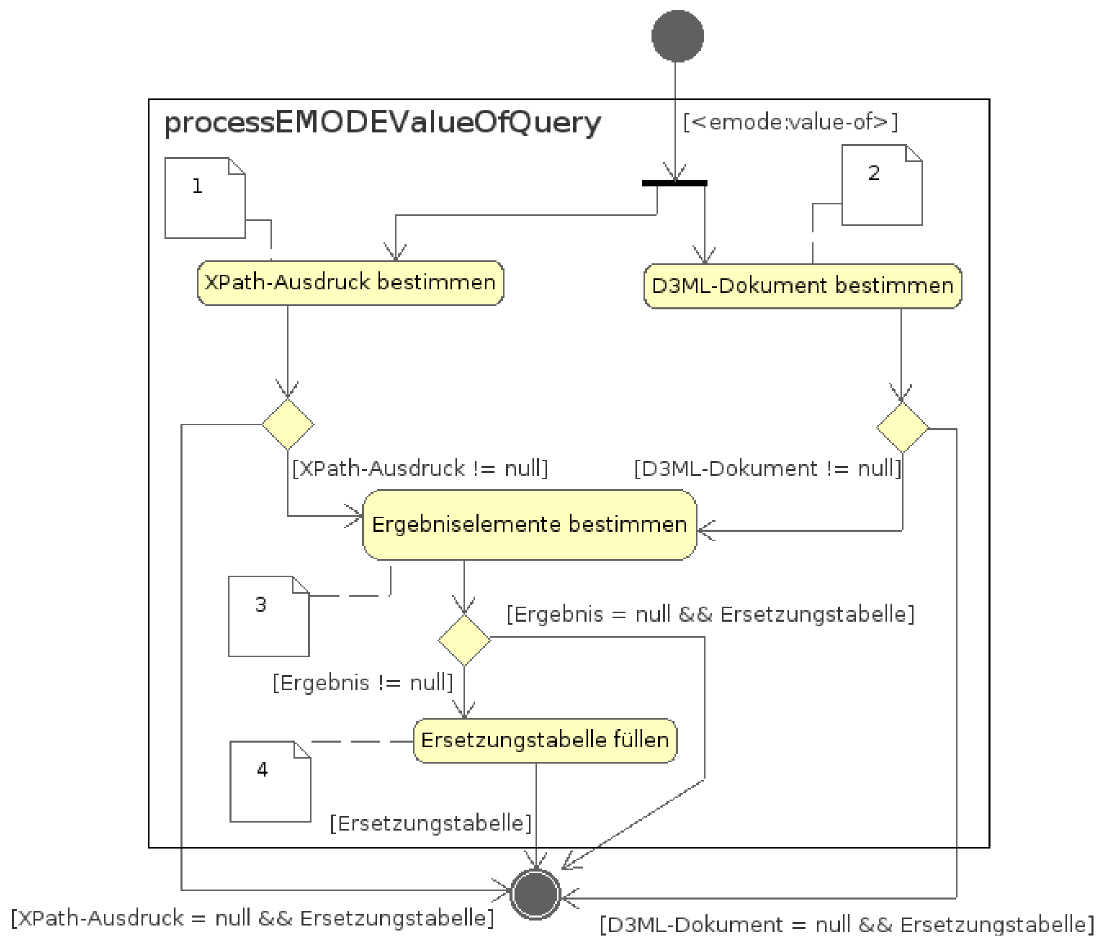


Abbildung 4.3.: processEMODEValueOfQuery(...) Methode

4.3.2. Die Verarbeitung der XForms-Aktionen

Die Menge der XForms-Aktionen besteht aus den Elementen `<xforms:setvalue>` und `<xforms:insert>` (siehe Tabelle 3.2 und Abb. 4.1).

Im Vergleich zu den EMODE-Aktionen muss die Verarbeitung der verschiedenen XForms-Aktionen nicht getrennt werden. Beide Elemente dienen ausschließlich der Manipulation der multimodalen Benutzerschnittstelle und erzeugen keine Resultate, die in den Strom der Eingabedaten zurückgeführt werden müssen.

Innerhalb der Methode `processXFormsActions(...)` besteht der Verarbeitungsprozess der XForms-Aktionen hauptsächlich aus der Transformation der XForms-Elemente in äquivalente *XUpdate*-Beschreibungen und der Modifikation der jeweiligen D3ML-Dokumente mit Hilfe der *XUpdate*-Anweisungen (siehe dazu Abb. A.5.4.1).

Jede XForms-Aktion, die in der Eingabenachricht enthalten ist, wird einzeln verarbeitet (Abb. A.5.4.1 Punkt 2). Das zu modifizierende D3ML-Dokument der multimodalen Benutzerschnittstelle wird durch das `xlink:href`-Attribut der XLINK-Spezifikation [DMO01] angegeben (Abb. A.5.4.1 Punkt 4).

Die MSC besitzt eine XSLT-Datei, die die Transformation der XForms-Aktionen in *XUpdate*-Anweisungen beschreibt. Die Umwandlung der XForms-Aktion wird mit Hilfe dieser Datei vollzogen (Abb. A.5.4.1 Punkt 3).

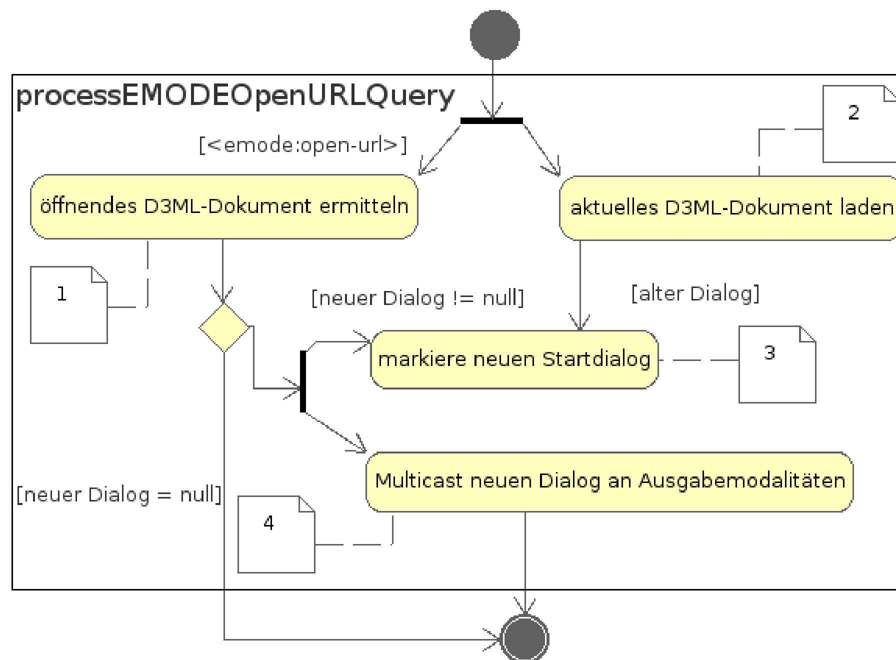


Abbildung 4.4.: processEMODEOpenURLQuery(...) Methode

Nachdem die *XUpdate*-Modifikation und das betreffende D3ML-Dokument zur Verfügung stehen, werden sie an das Objekt, welches die Veränderungen durchführen soll (`D3MLDBManager`), übergeben (Abb. A.5.4.1 Punkt 5).

Die Veränderungen der multimodalen Benutzerschnittstelle müssen, genauso wie in Abschnitt 4.2.1 *ModalityOutput*, an alle anderen Modalitäten übergeben werden, dementsprechend wird auch hier ein *Multicast* durchgeführt (Abb. A.5.4.1 Punkt 7). Dabei werden nicht die *XUpdate*-Modifikationen, sondern die XForms-Aktionselemente versendet. Dies hat folgende Ursache.

Da jede Ausgabemodalität ihr eigenes Ausgabeformat besitzt, kann keine allgemeingültige *XUpdate*-Beschreibung zur Modifikation des *User Interface*'s der Ausgabemodalität eingesetzt werden. Die XForms-Aktionen, wie z. B. `<xforms:setvalue>`, beschreiben die Modifikation von UI-Elementen. Soll z. B. der Wert eines Textfeldes in HTML verändert werden, so kann dies mit Hilfe spezieller HTML-Attribute erreicht werden. Dies bedeutet im Fall der XForms-Aktion `<xforms:setvalue>`, dass sie auf der Seite der Ausgabemodalität wiederum in eine, speziell für HTML angepasste, *XUpdate*-Beschreibung überführt werden muss. Deshalb ist es sinnvoller die XForms-Aktionen an die Ausgabemodalitäten zu versenden, anstelle der *XUpdate*-Modifikationsbeschreibung der D3ML-Dokumente.

Die XForms-Aktionen beziehen sich auf die Datenmodellelemente der EMODE Anwendung, welche im D3ML-Dokument enthalten sind. Damit die Aktionsbeschreibungen an die Ausgabemodalitäten übergeben werden können, gilt auch hier wie in Abschnitt 4.2.1 *ModalityOutput*, dass die Bindungen zwischen Datenmodellelementen und UI-Elementen aufgelöst werden müssen. Dies geschieht innerhalb der `prepareXFormsAction(...)`-Methode (Abb. A.5.4.1 Punkt 6).

Im letzten Schritt werden die Aktionsbeschreibungen in eine Liste eingefügt, die dem Entfernen der XForms-Aktionen aus der Eingabenachricht dienen soll (Abb. A.5.4.1 Punkt 8). Alle XForms-Aktionen werden am Ende der Eingabeanalyse komplett aus der Eingabenachricht entfernt. Die Abbildung A.5.3 soll den Prozess des Entfernens der XForms-Aktionselemente veranschaulichen.

Der Prozess der Analyse der Eingabedaten lässt sich durch folgende Teilschritte zusammenfassen (siehe dazu Abb. A.5.1).

1. Ermittlung der Aktionselemente und Einteilung nach der Kategorie (XForms- und EMODE-Aktionen)
2. Verarbeitung der Aktionselemente je nach Kategorie
3. Rückführung der Ergebnisse in den Datenstrom der Eingabe bzw. Entfernung der Aktionselemente
4. Übermittlung der resultierenden Eingabedaten an die EMODE Anwendung

Der Datenstrom der Eingabe kann nur noch aus Kommandos und zugehörigen Argumenten bestehen, die zum Aufruf der verschiedenen Anwendungsfunktionen dienen. Im nächsten Abschnitt wird der Prozess anhand einer Beispielanwendung erklärt.

4.4. Der Aufruf der Anwendungsfunktionen durch Kommandos und Reflektion

Nach dem Entwurf sind die EMODE Anwendungen für den Aufruf der Applikationsfunktionen selbst zuständig (siehe Abschnitt 3.3 Variante 2). Sie besitzen spezielle Komponenten, die auf der einen Seite den Funktionsaufruf durchführen und auf der anderen Seite die Ausgabeinformationen an die MSC versenden.

Der Funktionsaufruf wird durch das Objekt des `Commander`'s implementiert. Bevor dies jedoch geschehen kann, müssen die Kommandos und deren Argumente aus dem Strom der Eingabedaten extrahiert werden. Diese Aufgabe wird durch den `EMMAInterpreter` übernommen.

Die Kommandos und deren Argumente sind auf die Funktionen der Anwendung abgebildet. Die Abbildung wird durch eine externe Datei beschrieben und durch die Anwendung eingelesen (siehe Abschnitt 3.3.3.3).

Gelangt eine Nachricht von der Eingabemodalität über den Anwendungscontainer bis hin zur EMODE Anwendung, so wird sie zunächst vom `EMODEEndpoint` empfangen und anschließend an das Objekt des `EMMAInterpreter` übergeben. Im folgenden Abschnitt wird die Ermittlung der Kommandos und zugehöriger Argumente, aus dem Strom der Eingabedaten im EMMA-Format, erläutert.

4.4.1. Die Ermittlung der Kommandos und Argumente

Der `EMMAInterpreter` hat das Ziel, die Kommandos und die Argumente aus den Eingabedaten zu bestimmen. Die Eingabedaten liegen im EMMA-Format vor, ein Beispiel dafür ist das Listing 3.10 des Abschnitts 3.3.3.3. Sie werden in der Beispielanwendung mit Hilfe von *REXML*, eine XML-API für die Skriptsprache *Ruby*, in ein Datenstrukturbaum (vergleichbar mit DOM) überführt. Dies erleichtert das Lesen und Schreiben von XML-Dokumenten bzw. der EMMA-Eingabedaten.

Mit Hilfe der XPath-Technologie lassen sich der Kommandoname und die Argumentenmenge relativ unkompliziert bestimmen (Listing 4.5 Zeile 6; 9). Der Kommandoname ist im `<action>`-Element enthalten (siehe Listing 3.9) und muss ausgelesen werden (Listing 4.5 Zeile 13). Die Argumentenmenge wird durch eine Liste von Elementen repräsentiert.

Damit der Funktionsaufruf anhand des Kommandonamen und der Argumentenelemente erfolgreich durchgeführt werden kann, muss die Liste der Argumente, entsprechend der Methodensignatur, geordnet sein. Die Aufgabe wird durch die Methode `argumentsOfAction(...)` erfüllt (Listing 4.5 Zeile 16). Sie erhält den Kommandonamen, die ungeordnete Liste der Argumentenelemente und die Abbildungstabelle (Kommando - Funktion - Bindung) als Eingabeparameter. Die Tabelle ordnet jedem Kommandonamen einen Eintrag zu, der eine Anwendungsfunktion repräsentiert. Er enthält den Namen der Anwendungsfunktion und eine leere Argumententabelle.

Die Methodensignaturen der Applikationsfunktionen sind in der Datei der Abbildungsbeschreibung enthalten (siehe Abschnitt 3.3.3.3 Listing 3.9). Ein wichtiger Aspekt innerhalb der Abbildungsbeschreibung ist, dass die Namen der Argumente von Anfang an feststehen müssen. Der Anwendungsentwickler muss darauf achten, dass die Argumentenbezeichnungen innerhalb der D3ML-Dateien und der Abbildungsbeschreibung identisch sind.

Die Methode `argumentsOfAction(...)` ermittelt anhand des eindeutigen Argumentennamen (innerhalb des Kommando) den zugehörigen Wert und fügt ihn in die leere Argumententabelle ein. Die Argumententabelle sorgt dafür, dass die Ordnung der Argumente, entsprechend der Methodensignatur, erhalten bleibt.

Listing 4.5: EMMAInterpreter `checkForCommands(...)`-Methode

```
1  def checkForCommands(message, cmd2function)
2    begin
3      cmd = REXML::Document.new(message)
4  
5      # get action element
6      action_name = REXML::XPath.first(cmd, "//command/action")
7  
8      # get all arguments
9      arguments = REXML::XPath.match(cmd, "//command/*[name()!='action']")
10 
11     if action_name != nil then
12       # get action name
13       action_name = action_name.get_text().to_s()
14 
15       # get argument values
16       args = argumentsOfAction(arguments, action_name, cmd2function)
17       return MyCommand.new(action_name, args)
18     end
19   rescue Exception => e:
20     # exception handling
21     ...
22   end
23   return nil
24 end
```

Nachdem die Argumentenwerte in der richtigen Reihenfolge bestimmt wurden, werden sie als Liste zurückgegeben und anschließend in eine Kommandodatenstruktur integriert (Listing 4.5 Zeile 17). Sie wird zum Aufruf der entsprechenden Anwendungsfunktion durch den `Commander` benötigt.

4.4.2. Vom Kommando zur Anwendungsfunktion

Das Objekt des `Commander`'s implementiert den Funktionsaufruf mittels des sog. Reflektionsmechanismus von Ruby [TFH01].

Zunächst muss der Name der aufzurufenden Methode feststehen. Er ist in der Datei der Abbildungsbeschreibung enthalten und somit auch in der Abbildungstabelle (Kommando-Funktion-Bindung). Mit Hilfe des zugeordneten Kommandonamen kann die Methode `fetchEntry(...)` den Eintrag der Anwendungsfunktion aus der Abbildungstabelle (`@cmd2function` - Instanzattribut der Klasse `Commander`) entnehmen und somit den internen Methodenbezeichner bereitstellen (Listing B.4 Zeile 9).

Die zum Aufruf benötigte Argumentenliste steht durch den Eingabeparameter `arguments` zur Verfügung.

Damit der Funktionsaufruf ohne Probleme durchgeführt werden kann, muss überprüft werden, ob eine Methode mit dem gegebenen Methodenbezeichner existiert. Ist dies der Fall, so muss ebenfalls die Anzahl der benötigten Argumente mit der Anzahl der bereitgestellten Argumente verglichen werden (Listing B.4 Zeile 15-17). Können keine Unregelmäßigkeiten festgestellt werden, so wird die Methode aufgerufen(Listing B.4 Zeile 20). Dies erfolgt über das eigentliche Applikationsobjekt `application`. Es enthält alle Anwendungsfunktionen, die von außen aufgerufen werden können. Im Beispiel einer *Chat*-Anwendung enthält das Objekt die Methode zum Versenden von Nachrichten. Bei komplexen Anwendung kann das Objekt als sog. *Wrapper* fungieren, der weitere Teilprozesse auslöst.

In Ruby wird der dynamische Methodenaufruf mit Hilfe der `send(...)`-Methode durchgeführt. Jede Methode jedes Objektes kann durch sie indirekt aufgerufen werden. In anderen reflektiven Programmiersprachen existieren ähnliche Funktionen, in Java ist es die `invoke(...)`-Methode der `java.lang.reflect.Method`-Klasse.

Durch den dynamischen Methodenaufruf ist es möglich, anhand des empfangenen Eingabekommandos zu entscheiden welche Funktion ausgeführt werden soll. Er bildet die Grundlage für den Entwurf der EMODE Anwendung nach der 2. Variante des Abschnitts 3.3.

5. Zusammenfassung

Das Ziel der Arbeit war die Erstellung einer flexiblen, plattformunabhängigen und erweiterbaren Laufzeitumgebung nach der EMODE Spezifikation [HHN].

Die Eigenschaft der Flexibilität und der Erweiterbarkeit wird mit Hilfe des *OSGi-Frameworks* erreicht. Es bildet das Fundament der MSC. Alle Modalitäten müssen als sog. *OSGI-Bundle* implementiert werden. Sie können durch das *Framework* zur Laufzeit hinzugefügt und auch wieder entfernt werden.

Die Plattformunabhängigkeit wird ebenfalls mit Hilfe des *OSGi-Frameworks* realisiert. Da es in der Programmiersprache Java implementiert ist, kann es in einer heterogenen Umgebung eingesetzt werden, z.B. auf mobilen Endgeräten.

Die Modalitäten repräsentieren die verschiedenen Ein- und Ausgabegeräte auf der Ebene der Software. Sie sind jedoch nicht nur auf Ein- und Ausgabegeräte beschränkt, sondern können jede Art von Eingabequelle und Ausgabemöglichkeit beinhalten. Eine Eingabemodalität kann z.B. den Empfang von GPS-Koordinaten realisieren und an eine Navigationsanwendung übergeben. Die Anwendungsbereiche werden durch das *OSGi-Framework* nicht begrenzt.

Die Modalitäten werden in dieser Arbeit in folgende Kategorien unterschieden.

- Eingabemodalität
- Ausgabemodalität
- Ein-/Ausgabemodalität

Da die Eingabemodalitäten verschiedene Eingabegeräte repräsentieren und diese unterschiedliche Eingabeinformationen erzeugen, wird in dieser Arbeit das EMMA-Format zur Überbrückung der Unterschiede eingesetzt. Es enthält die Eingabeinformationen in einer einheitlichen interpretierten Form. Jede Eingabemodaltität erzeugt somit einen Eingabedatenstrom im EMMA-Format, der durch die MSC an eine EMODE Anwendung weitergeleitet wird.

Die Ausgabemodalitäten haben die Aufgabe die Ausgabedialoge einer Anwendung durch die verschiedenen Ausgabegeräte darzustellen. Die Anwendungsdialoge der multimodalen Benutzerschnittstelle werden deklarativ durch D3ML-Dokumente beschrieben. Sie enthalten das *Layout* und die Elemente des *User Interface*. Zusätzlich beschreiben sie die Bindung von UI-Element und Datenmodellelement. Dadurch wird bestimmt, welche Information durch welches UI-Element dargestellt werden soll.

Jede Ausgabemodalität muss die Dialoge der multimodalen Benutzerschnittstelle soweit an das jeweilige Ausgabegerät anpassen, so dass die Informationen der Dialoge verständlich ausgegeben werden können.

Die MSC ist der Vermittler zwischen den EMODE Anwendungen und den Modalitäten. Sie ist so konzipiert, dass mehrere Anwendungen gleichzeitig verwendet werden können. Dies wird mit Hilfe der Anwendungscontainer erreicht. Jeder Container enthält die Ressourcen der Anwendung und regelt die Kommunikation mit den Modalitäten.

Aus der Sicht der Anwendung ist eine Modalität eine Komponente, die Ein- bzw. Ausgabedienste bereitstellt. Die Modalitäten besitzen nach dem Konzept der SOA Servicebeschreibungen, die in Meta-Informationsdateien enthalten sind. Jeder Anwendungscontainer kann auf die Meta-Informationen der Modalitäten zugreifen und entscheiden ob sie den Anforderungen der EMODE Anwendung gerecht werden.

Wird eine Modalität durch eine EMODE Anwendung genutzt, dann steht sie nur dieser Anwendung zur Verfügung. Der Anwendungscontainer besitzt die Modalität solange bis der Anwender sie deaktiviert oder die Anwendung beendet. Die Anwendungen können keine Modalitäten untereinander austauschen. Die Anwendungscontainer existieren unabhängig voneinander, d. h. es findet kein Datenaustausch zwischen den Anwendungen statt.

Die Verteilung der Modalitäten, auf die EMODE Anwendungen, ist simpel. Die Anwendung, die als erste registriert wird, erhält alle Modalitäten aus der Menge der momentan Verfügbaren, die den Anforderungen der Anwendung entsprechen. Ist einer Anwendung bereits eine Modalität zugeordnet, so kann keine weitere Modalität an den Anwendungscontainer übergeben werden, die die gleichen Eigenschaften besitzt. Bestimmte Anforderungen der Anwendung können immer nur durch eine Modalität erbracht werden. Dadurch kann eine alternative Modalität, die die selben Anforderungen wie die aktuelle Modalität erfüllt, z. B. die Benutzung des selben Eingabegerätes, erst nach deren Deaktivierung verwendet werden.

5.1. Vergleich der EMODE MSC mit dem JANUS-System

In diesem Abschnitt soll die MSC nach der EMODE Spezifikation mit dem Janus-System verglichen werden. Das Janus-System ist eine multimodale, komponenten-basierte und serviceorientierte *Middleware*, die für den Einsatz in automotiven Umgebungen konzipiert wurde (siehe [Emb04]). Durch den Vergleich wird die MSC zusammenfassend charakterisiert.

Die Gemeinsamkeit beider Systeme kennzeichnet sich durch folgende Punkte.

- Beide Systeme verwenden das *OSGi-Framework* zur Realisierung der Modalitätenverteilung und deren Lebenszyklus.
- Die Modalitäten werden als Repräsentation der Ein- und Ausgabegeräte auf der Ebene der Software angesehen.
- Die Modalitäten werden durch Meta-Informationen charakterisiert.
- Eine Anwendung kann mehrere Modalitäten zur gleichen Zeit verwenden.

Die folgende Tabelle 5.1 soll beide System voneinander unterscheiden.

Die wesentlichen Unterschiede beider Systeme liegen im Bereich der Verwaltung parallel ablaufender Anwendungen und im Bereich der Kommunikation.

Die MSC betrachtet alle EMODE Anwendungen als eigenständig ablaufende Prozesse. Die Kommunikationen erfolgt durch Interprozeßkommunikation. Der Anwendungscontainer fungiert als Stellvertreter der Anwendung innerhalb der MSC. Das Ziel des Entwurfs liegt in der breiten Unterstützung verschiedener Anwendungsimplementierungen. Durch die programmiersprachenunabhängigen Ein- und Ausgabeformate kann der Anwendungsentwickler die EMODE Applikationen flexibel und unabhängig von der MSC gestalten.

Eigenschaft	EMODE MSC	JANUS
Unterstützung paralleler Anwendung	Jede Applikation besitzt einen Anwendungscontainer.	Jede Applikation wird durch ein Bundle im OSGi-Framework implementiert.
Unterstützung paralleler Modalitäten	Einem Anwendungscontainer können mehrere Modalitäten zugeordnet sein. Eine Modalität ist zur Laufzeit einer Anwendung zugeordnet.	Die Modalitäten werden durch mehrere Anwendungen benutzt. Eine Modalität kann mehreren Anwendungen zugeordnet sein.
Unterstützung verteilter Modalitäten	nein	Mit Hilfe der JINI Technologie können auch verteilte Modalitäten genutzt werden.
Kommunikation	*Pipe*-ähnliche Kommunikation	Ereignis-gesteuerte Kommunikation, Kommunikation über Pipes
Austauschformat	EMMA als Eingabeinterpretation, XUpdate als Ausgabemodifikation	Java Ereignisse und Datenströme für Ein- und Ausgabe
Charakter der Modalitäten	generisch, unabhängig von der Anwendung, Ein- und Ausgabemodalitäten passen die Benutzerschnittstelle entsprechend der Ein- bzw. Ausgabegeräte an	nicht generisch, keine Anpassung
multimodale Benutzerschnittstelle	Jede Anwendung definiert die Benutzerschnittstelle deklarativ durch D3ML-Dokumente. Sie bilden in ihrer Gesamtheit die multimodale Benutzerschnittstelle.	nicht vorhanden
Unterstützung heterogener Anwendungen	Unterstützung heterogener Anwendung durch standardisierte Ein- und Ausgabeformate; Aufruf der Applikationsfunktionen durch die Anwendung selbst	keine direkte Unterstützung heterogener Anwendungen

Tabelle 5.1.: Unterschiede zwischen MSC und dem Janus-System

Im Janus-System werden alle Anwendungen als *OSGi-Bundle* implementiert. Jedes Bundle ist auf die Java-Technologie begrenzt. Nach diesem Entwurf muss jede heterogene Anwendung eine *OSGi-Bundle*-Implementierung besitzen. Das *Bundle* besitzt in diesem Fall eine *Wrapper*-Funktion. Der Anwendungsentwickler hat dadurch einen erhöhten Implementierungsaufwand.

Die Kommunikation zwischen den Modalitäten und den Anwendungen basiert im Janus-System hauptsächlich auf dem Austausch von Ereignissen. Die Ereignisse werden durch sog. *JanusEvents* [Emb04] repräsentiert. Es sind Java-Objekte, die die Eingabe von Modalitäten bzw. die Ausgabe der Anwendungen darstellen. Durch die starke Verknüpfung der Kommunikation zwischen den Modalitäten und den Anwendungen, mit der Java Technologie und dem *OSGi-Framework*, ist die Unterstützung heterogener Anwendungen zusätzlich erschwert.

5.2. Die „*Proof-of-Concept*“ – EMODE Applikation

In diesem Abschnitt wird die EMODE Applikation „myRubyTalk“ als sog. „*Proof-of-Concept*“ vorgestellt. Das Ziel der Entwicklung ist einerseits die Überprüfung der Machbarkeit der verschiedenen Konzepte, andererseits die Erforschung der Problemfelder, wie z.B. der Informationsaustausch über die Grenzen der unterschiedlichen Technologien.

Die EMODE Anwendung „myRubyTalk“ ist eine multimodale *Chat*-Anwendung, die in der Skriptsprache Ruby implementiert wurde. Sie ist vergleichbar mit ähnlichen Anwendungen, wie z. B. *ICQ-Clients*. Die Absicht dabei ist, zu zeigen, dass der Datenaustausch zwischen der MSC und der unabhängigen Anwendung, über die Grenzen der verschiedenen Implementationen, nach den vorgestellten Konzepten machbar ist.

Innerhalb eines Anwendungsszenarios kommunizieren mindestens zwei „myRubyTalk“ -*Clients* über einen *Server*. Der *Server* selbst ist eine eigenständige Anwendung der keine multimodale Funktionalität besitzt und nicht durch die MSC registriert wird. Er ist nur für die Kommunikation der beiden *Chat*-Applikationen zuständig, er führt die Nachrichtenweiterleitung und die Registrierung / Deregistrierung der Kommunikationspartner durch. Die MSC erkennt ausschließlich die *Client*-Anwendungen als EMODE Applikationen. Aus Testzwecken laufen der „myRubyTalk“ -*Server* und die *Clients* lokal auf dem selben *Host*.

Die MSC und die Modalitäten werden als *OSGi-Bundle* ausgeliefert und sollen mittels des *OSGi-Framework's* „Knopflerfish“ [Gat06] ausgeführt werden. Es bietet neben dem Basis-*Framework* weitere zusätzliche Dienste, bspw. das *Desktop*-Werkzeug oder HTTP-Dienste in Form eines *WebServer*'s, an.

Die Steuerung des Lebenszyklus der verschiedenen *OSGi-Bundle* erfolgt durch das Desktop-Werkzeug der „Knopflerfish“ -Umgebung (siehe Abb. 5.1).

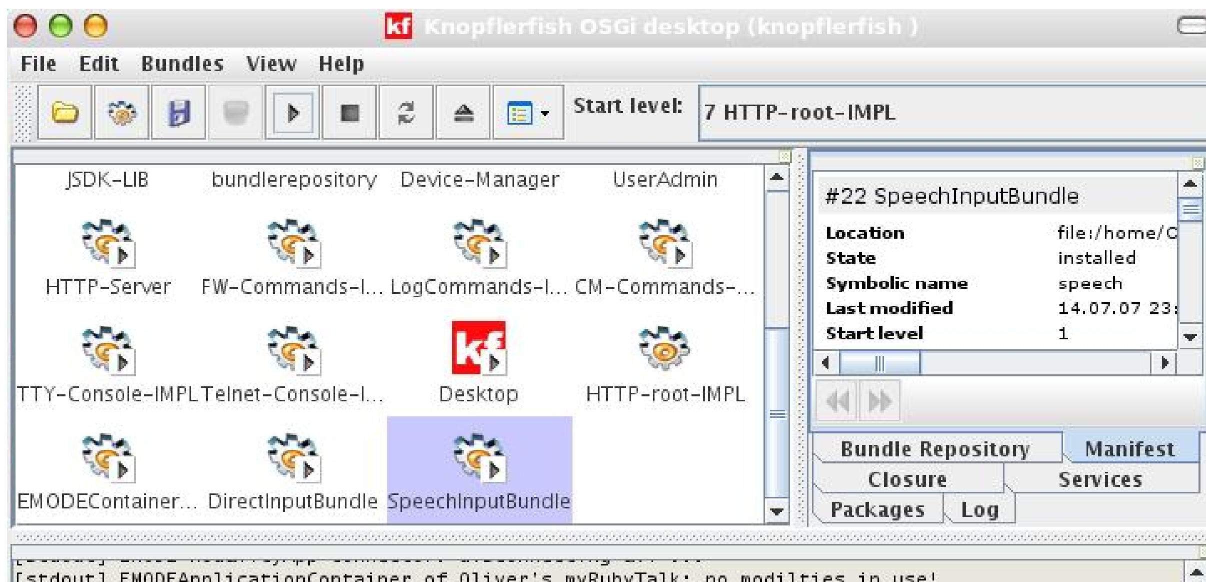


Abbildung 5.1.: Knopflerfish-Desktop

Im folgenden Abschnitt wird die *Chat*-Anwendung und deren multimodale Funktionen erläutert.

5.2.1. Die Chat-Anwendung

Die *Client*-Applikation soll in dieser Arbeit durch die Modalität der direkten Manipulation und durch die Spracheingabemodalität, wie sie in den Abschnitten 3.2.4 und 3.2.5 beschrieben sind, gesteuert werden. Die Ausgabe der Anwendung erfolgt visuell mit Hilfe eines HTML-Browsers durch die Modalität der direkten Manipulation (siehe Abschnitt 3.2.5). Die Modalität der Sprachsynthese erzeugt Ausgaben über die Lautsprecher.

Nachfolgend wird die Verzeichnisstruktur und der Anwendungsverlauf dokumentiert.

Die Verzeichnisstruktur

Die Verzeichnisstruktur der *Chat*-Anwendung wird durch die Tabelle 5.2 veranschaulicht.

Ordnername	Ordnerinhalte
d3ml	enthält alle D3ML-Dokumente der Anwendung
image	enthält alle Bilddateien der Anwendung
media	enthält alle Mediendateien der Anwendung, z.B. Audiodateien
resources	enthält alle Ressourcendateien, z.B. Meta-Informationen der Anwendung im Java Properties-Format

Tabelle 5.2.: Ordnerstruktur der „myRubyTalk“ Applikation

Alle Ordner die in der Tabelle 5.2 beschrieben werden, sind innerhalb des Wurzelverzeichnis der Anwendung zu finden. Im Fall der „myRubyTalk“ -*Chat* Applikation ist, neben den Ordnern, auch die ausführbare Anwendungsdatei, `client_X.rb` ('X' steht für die Nummer der Anwendung), enthalten.

Start der Anwendung

Bevor eine EMODE Anwendung gestartet werden kann, muss das *OSGi-Bundle* der MSC in die *OSGi*-Laufzeitumgebung integriert und aktiviert werden (siehe dazu [Gat06]). Nachdem dies geschehen ist, kann der *Chat*-Client gestartet werden. Die MSC registriert die Anwendung. In dem Fall, in dem die Modalitäten der Spracheingabe, Sprachsynthese und der direkten Manipulation bereits in die Laufzeitumgebung integriert wurden, starten alle Modalitäten sofort - unter der Voraussetzung, dass sie die Anforderungen der Anwendung erfüllen. Sind noch keine Modalitäten in die Laufzeitumgebung integriert und aktiviert worden, so können sie auch nach dem Start der Anwendung eingefügt werden.

Nachdem die Modalitäten und die EMODE Anwendung verknüpft wurden, werden sie durch den Anwendungscontainer innerhalb der MSC gestartet. Die Modalität der direkten Manipulation erzeugt aus der D3ML-Beschreibung des Startdialoges ein entsprechenden HTML-Dialog und gibt diesen mit Hilfe eines Browser aus (siehe Abb. 5.2)

Die Spracheingabemodalität gibt mit Hilfe des sog. `JSGFViewer` (Abb. 5.3) mögliche Eingabeelemente vor, die zum Zeitpunkt des aktuellen UI-Dialoges anwendbar sind. Dabei handelt es sich um die Spitznamen der möglichen Anwendungsnutzer und um das Kommando zur Bestätigung des Namens.

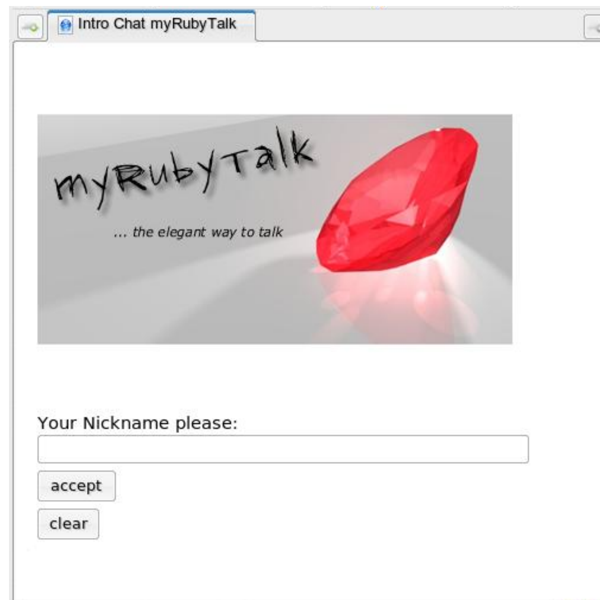


Abbildung 5.2.: Startdialog introUI.html

Abbildung 5.3.: JSGFViewer - Sprachkommandos während des Start-Dialoges

Die Kommunikation

Der Austausch der Textnachrichten erfolgt über den *MessageBrowser* des *Chat*-Dialoges (siehe Abb. 5.4). Die Nachrichten werden visuell und akustisch durch die entsprechenden Modalitäten dargestellt. Der Nutzer hat jederzeit die Möglichkeit, die akustische Ausgabe der neuen Nachrichten, durch das Desktop-Werkzeug der „Knopflerfish“ -Umgebung zu deaktivieren bzw. wieder zu reaktivieren.

Die Liste der verfügbaren Kommunikationspartner wird visuell durch die Modalität der direkten Manipulation (Abb. 5.4) und akustisch durch die Sprachsynthesemodalität ausgegeben. Die sprachliche Ausgabe der neuen Kommunikationspartner erfolgt in dieser Arbeit dann, wenn ein neuer Client in die Liste hinzugefügt oder wieder entfernt wird. Wird ein Client durch den Maus-Cursor selektiert, so erzeugt die Modalität der Sprachsynthese ebenfalls eine entsprechende akustische Darstellung der Auswahl.

Die Spracheingabemodalität besitzt auch zu diesem UI-Dialog eine bestimmte Menge von möglichen Eingabekommandos, die der `JSGFViewer` (Abb. 5.5) anzeigt. Neue Nachrichten können hauptsächlich durch die Modalität der direkten Manipulation erzeugt werden. Die Spracheingabemodalität bietet zusätzliche Textelemente an (Abb. 5.5). Um Nachrichten an entfernte Kommunikationspartner versenden zu können, kann der Nutzer den „Send“ -Knopf der visuellen Darstellung betätigen oder das Sprachkommando „write to client“ in das Mikrophon sprechen.

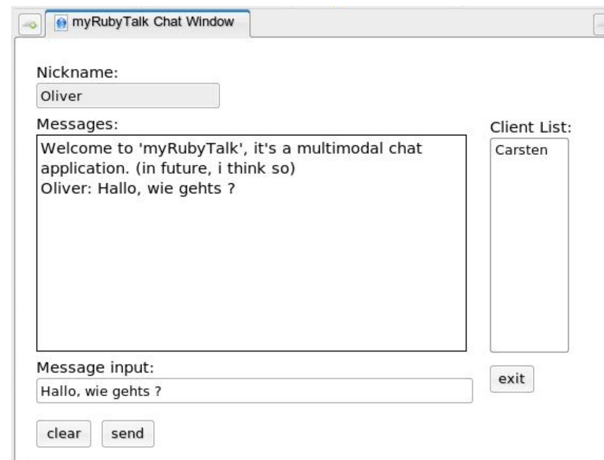


Abbildung 5.4.: MessageBrowser chatUI.d3ml

Abbildung 5.5.: JSGFViewer - Sprachkommandos während des *Chat*-Dialoges

5.3. Ausblick

Der Prototyp der EMODE MSC erfüllt nicht alle Funktionen, die in [HHN] spezifiziert werden. Die folgenden Punkte stellen mögliche Erweiterungen dar.

Die Vereinigung der Eingabedaten

In dieser Arbeit wurde die Vereinigung der Eingabedaten nicht betrachtet. Sie ist jedoch notwendig, um die Präzision der Eingabeverarbeitung zu verbessern bzw. um neue Eingabemethoden zu ermöglichen. Ein Beispiel dafür ist die Kombination der Spracheingabe und die Modalität der direkten Manipulation. Mit Hilfe der Maus können Textbereiche ausgewählt werden, ein Stimmenkommando sorgt für das Löschen dieser Bereiche. Die Vereinigung der Eingabedaten muss innerhalb des Anwendungscontainers durchgeführt werden, in ihm treffen alle Eingabedatenströme zusammen.

Die Verteilung der Ausgabeinformationen anhand zeitlicher Beziehungen

Die Aufteilung der Ausgabedaten findet nach dem bisherigen Entwicklungsstand ohne Berücksichtigung von zeitlichen Beziehungen statt. Die Ausgabeinformationen werden alle gleichzeitig an die Modalitäten verteilt. Die zeitlichen Beziehungen werden mit Hilfe von Zeitstempeln angegeben. Sie werden in die Ausgabenachrichtenelemente integriert, z.B. durch ein spezielles Attribut,

welches in die XUpdate-Anweisungen der EMODE Anwendungen integriert wird. Die MSC muss die Ausgabe anhand der Zeitstempel koordinieren und verteilen (siehe [HHN]).

Die direkte Ansteuerung einzelner Ausgabemodalitäten

Einer EMODE Anwendung können mehrere Ausgabemodalitäten zugeordnet sein, sie kann bisher nicht die Informationen direkt an einzelne Modalitäten übergeben. Das folgende Szenario soll als Beispiel dienen. Eine Anwendung möchte eine Grafik allein durch die visuelle Ausgabemodalität darstellen. Ein Text dazu soll allein durch die Modalität der Sprachsynthese vorgelesen werden. Um dies zu realisieren müssen die XUpdate-Anweisungen die beabsichtigte Ausgabemodalität anzeigen. Die MSC kann anhand der speziellen Informationen, in den XUpdate-Anweisungen, erkennen wohin die Ausgabenachricht geschickt werden soll. Eine Möglichkeit die Modalitäten in den XUpdate-Anweisungen anzuzeigen besteht darin, Attribute zu verwenden, welche die bevorzugte Ausgabemodalität beschreiben (ähnlich der Anforderungsbeschreibungen der EMODE Anwendungen).

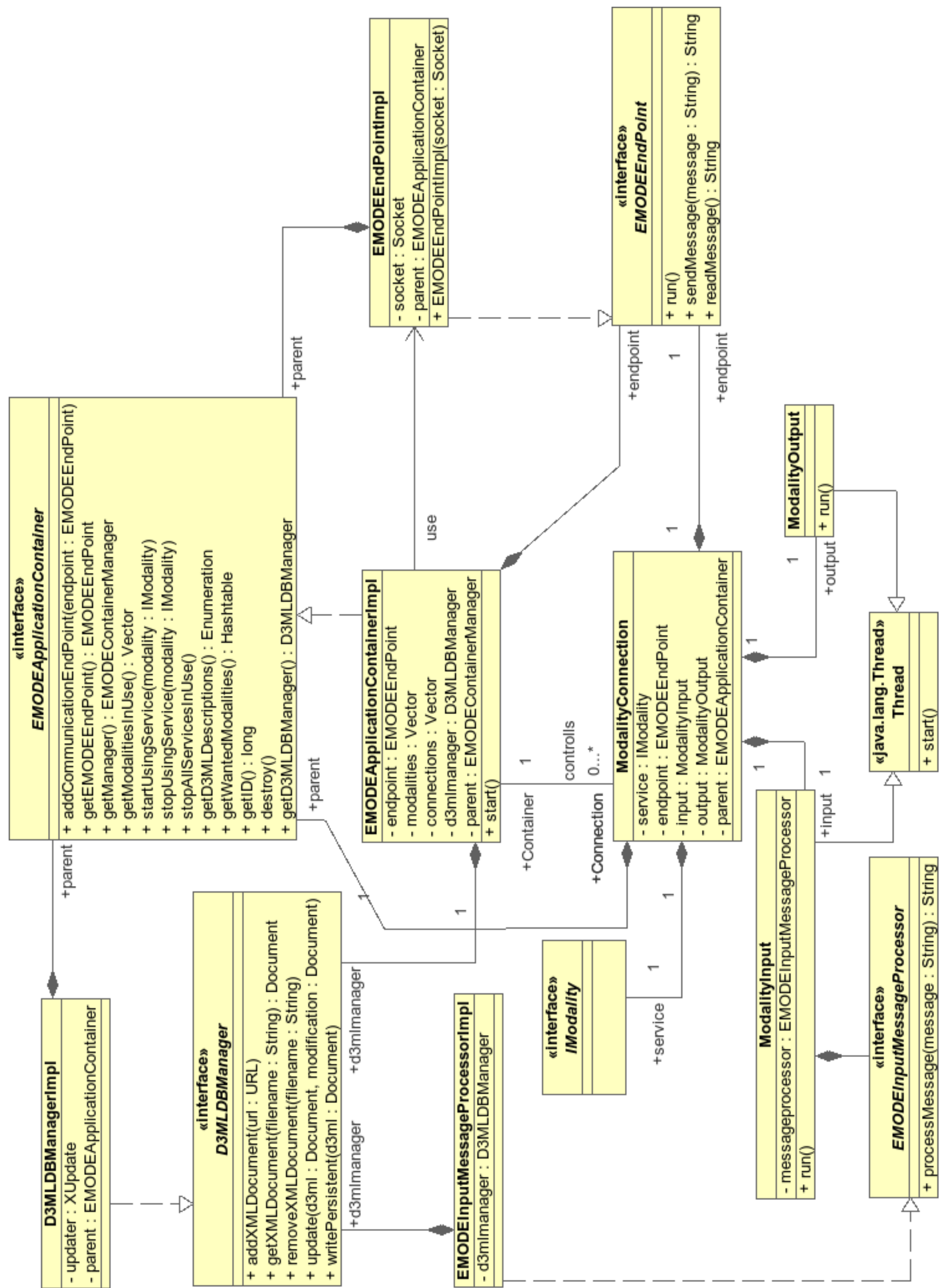
Wahrnehmung des Anwendungskontext durch Sensoren

Eine zusätzliche Erweiterung der MSC sind Sensorkomponenten, die die verschiedenen Umweltbedingungen wahrnehmen. Der Einsatz der Umweltsensoren kann die Bedienung von mobilen Endgeräten in den verschiedenen Situationen vereinfachen. Die Komponenten können auf der Basis des *OSGi-Frameworks* als *Plug-In* entwickelt werden. Die Implementierungen der Umweltsensoren würde neben den Modaltitäten als *OSGi-Bundles* zur Verfügung stehen. Die MSC müsste eine zusätzliche Funktion besitzen, um die verschiedenen Informationen der Umweltsensoren auszuwerten. Mit Hilfe der Informationen kann die MSC entscheiden welche Modalität im aktuellen Anwendungskontext besser zur Ein- oder Ausgabeverarbeitung geeignet ist. Zusätzlich muss die MSC einen Mechanismus besitzen, um die unterschiedlichen Entscheidungskriterien aufnehmen zu können. Eine Möglichkeit besteht darin, dass die Meta-Informationen der Modalitäten durch die Angabe der verschiedenen Anwendungskriterien erweitert werden. Ein Beispiel dafür ist die Spracheingabemodalität, deren Meta-Informationen angeben, ab welcher Umgebungslautstärke die Modalität zur Verfügung steht. Die MSC muss zur Laufzeit die Daten des Lautstärkesensors mit der oberen und unteren Grenze des zulässigen Lautstärkebereichs vergleichen und entscheiden ob der Einsatz der Modalität sinnvoll ist.

A. UML Diagramme

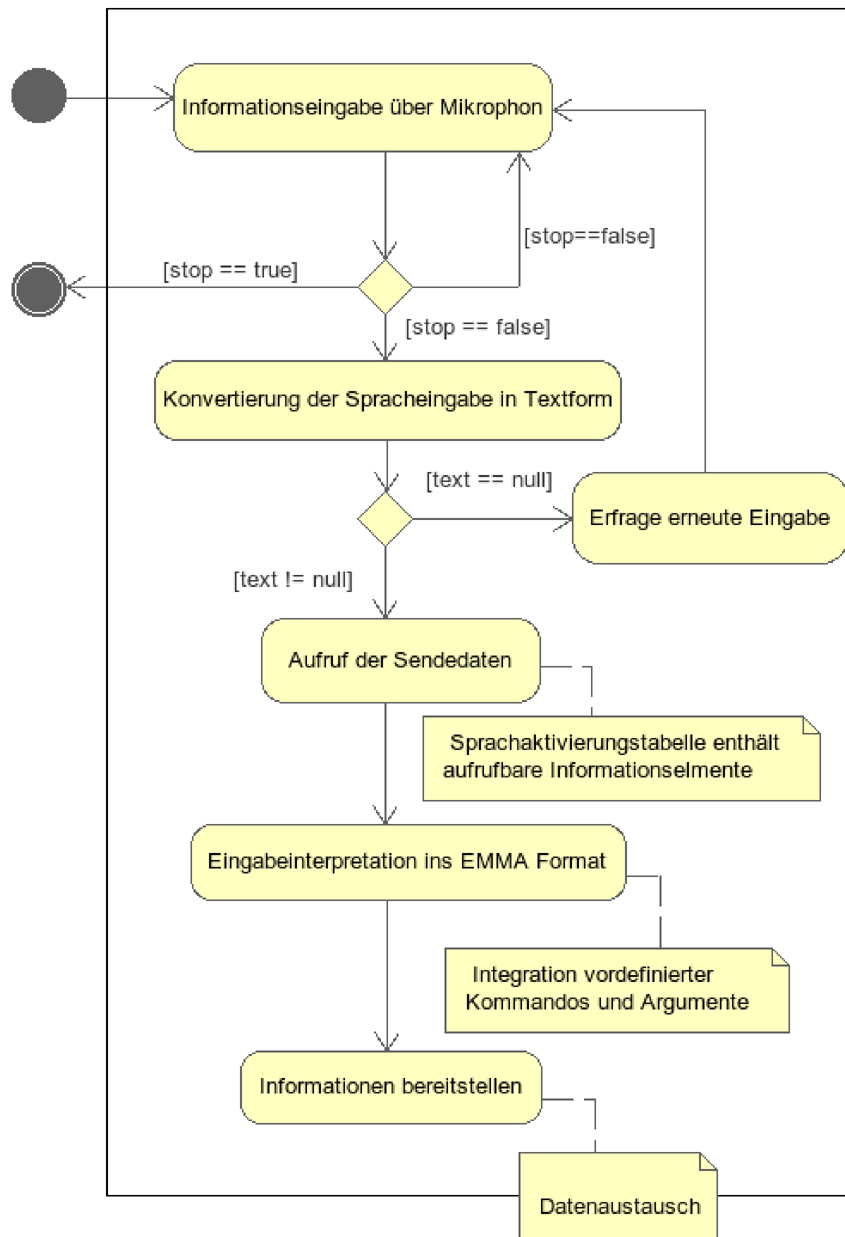
A.1. MSC Klassen

A.2. Anwendungscontainer

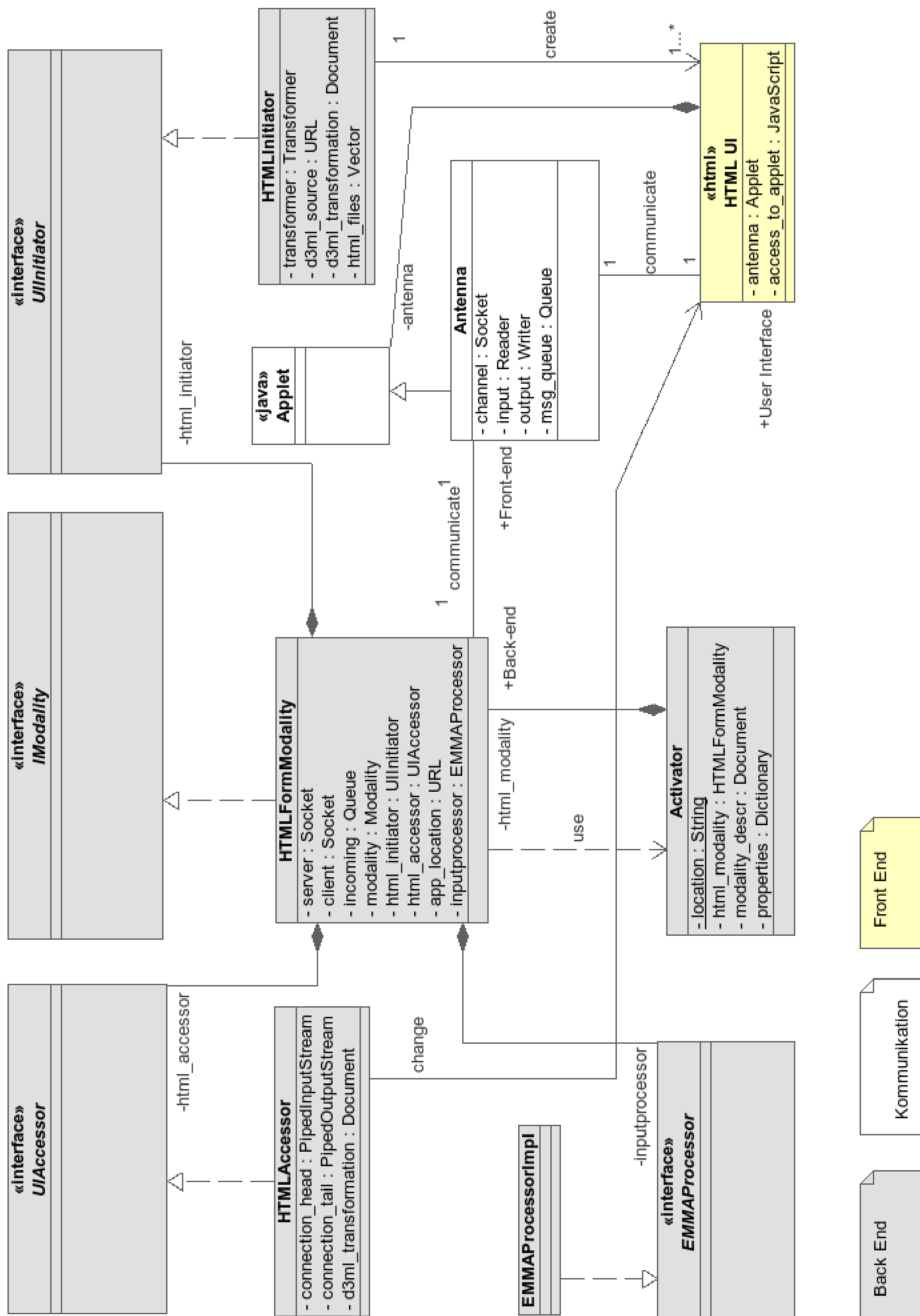


A.3. Modalitäten

A.3.1. Aktivitäten der Spracheingabemodalität

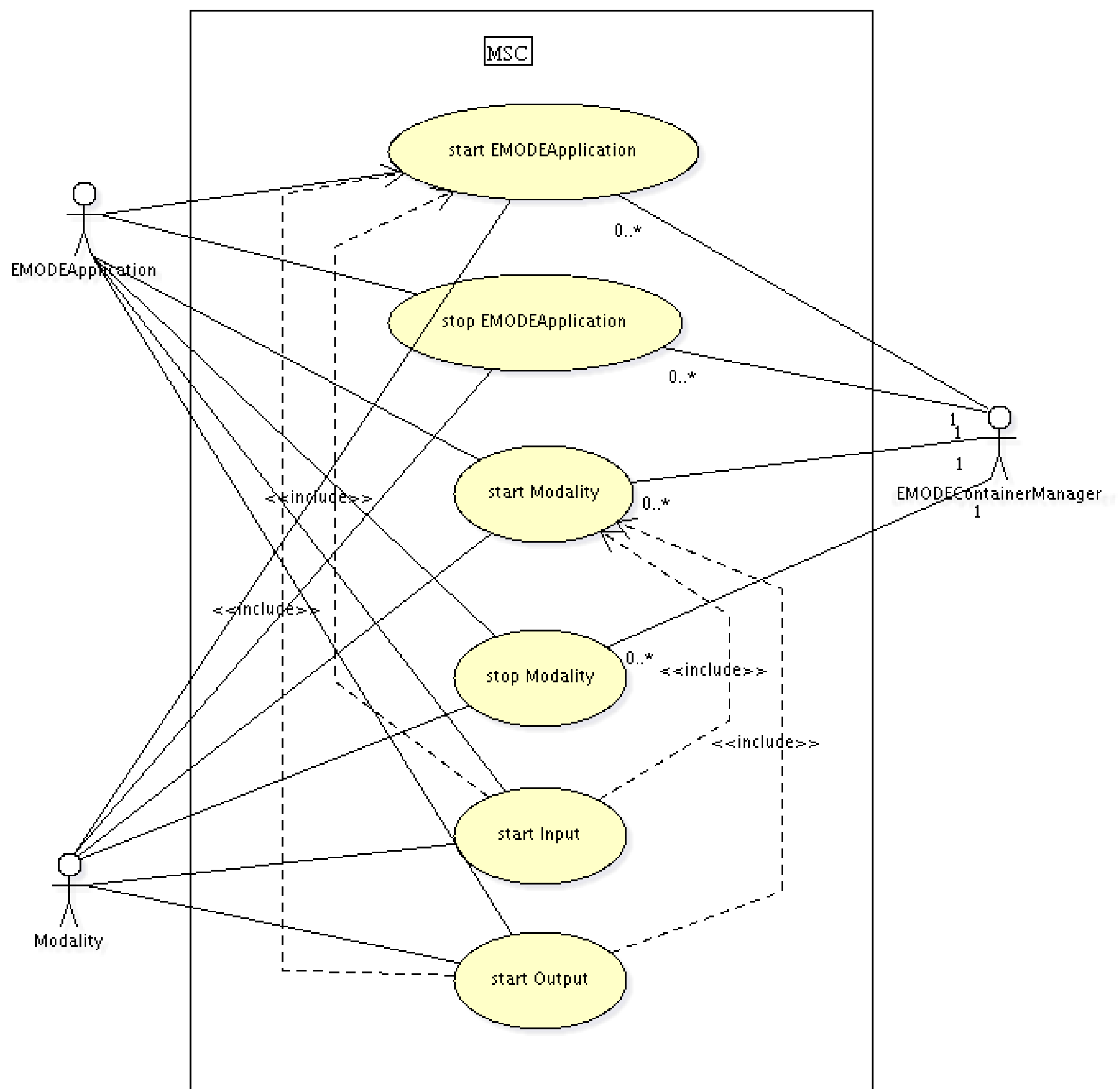


A.3.2. Entwurf der Modalität der direkten Manipulation

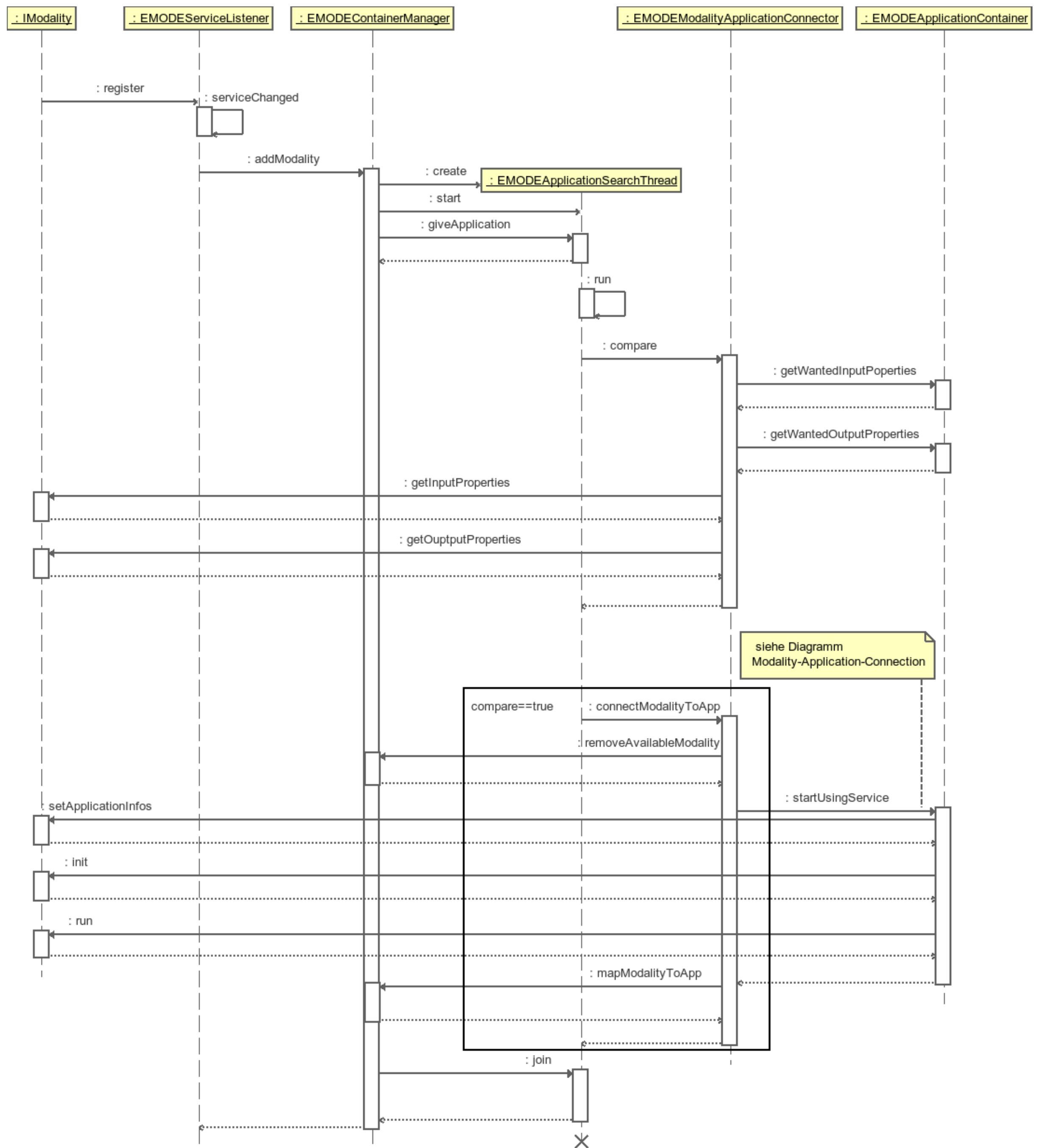


A.4. Anwendungsfälle

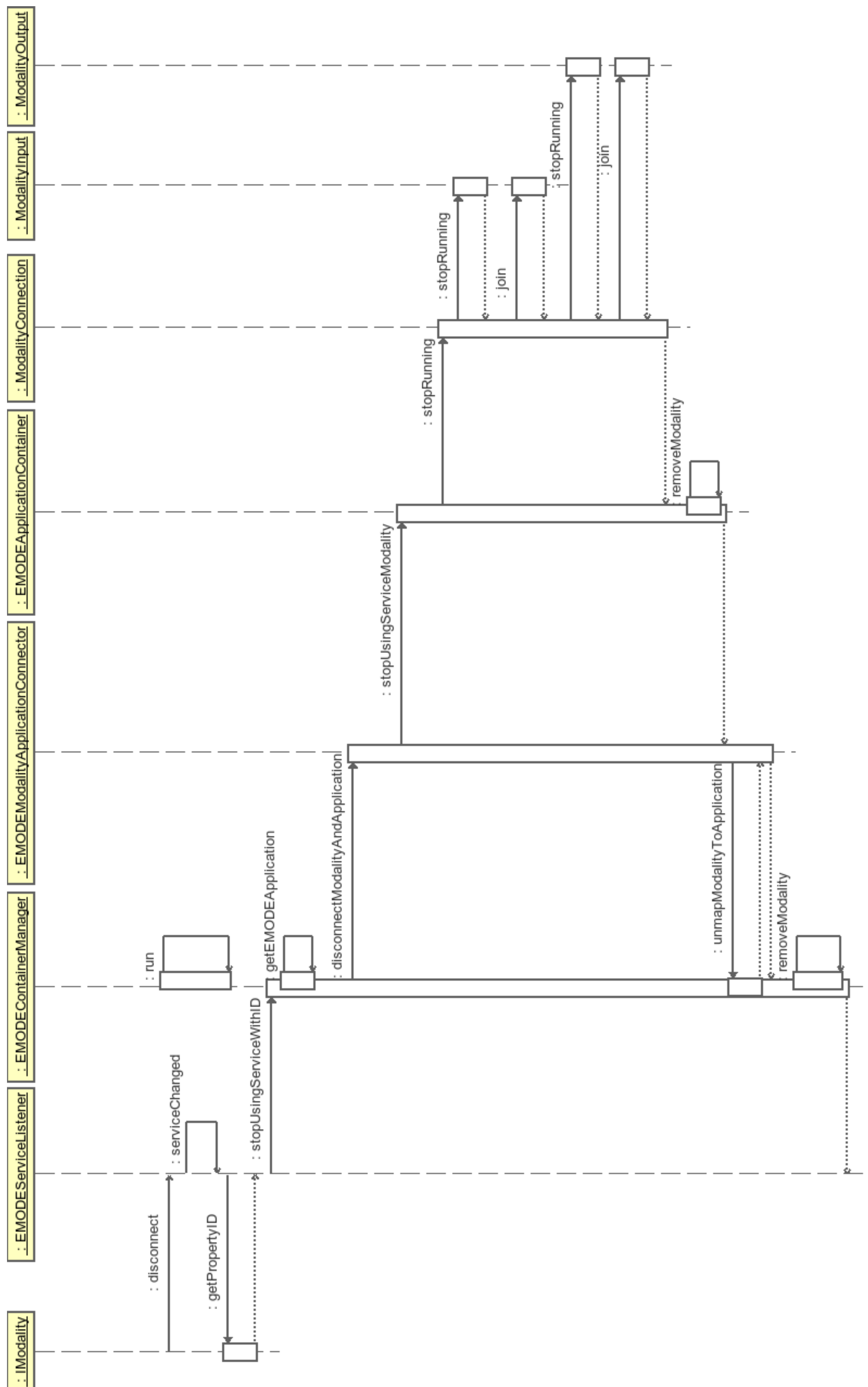
A.4.1. Überblick



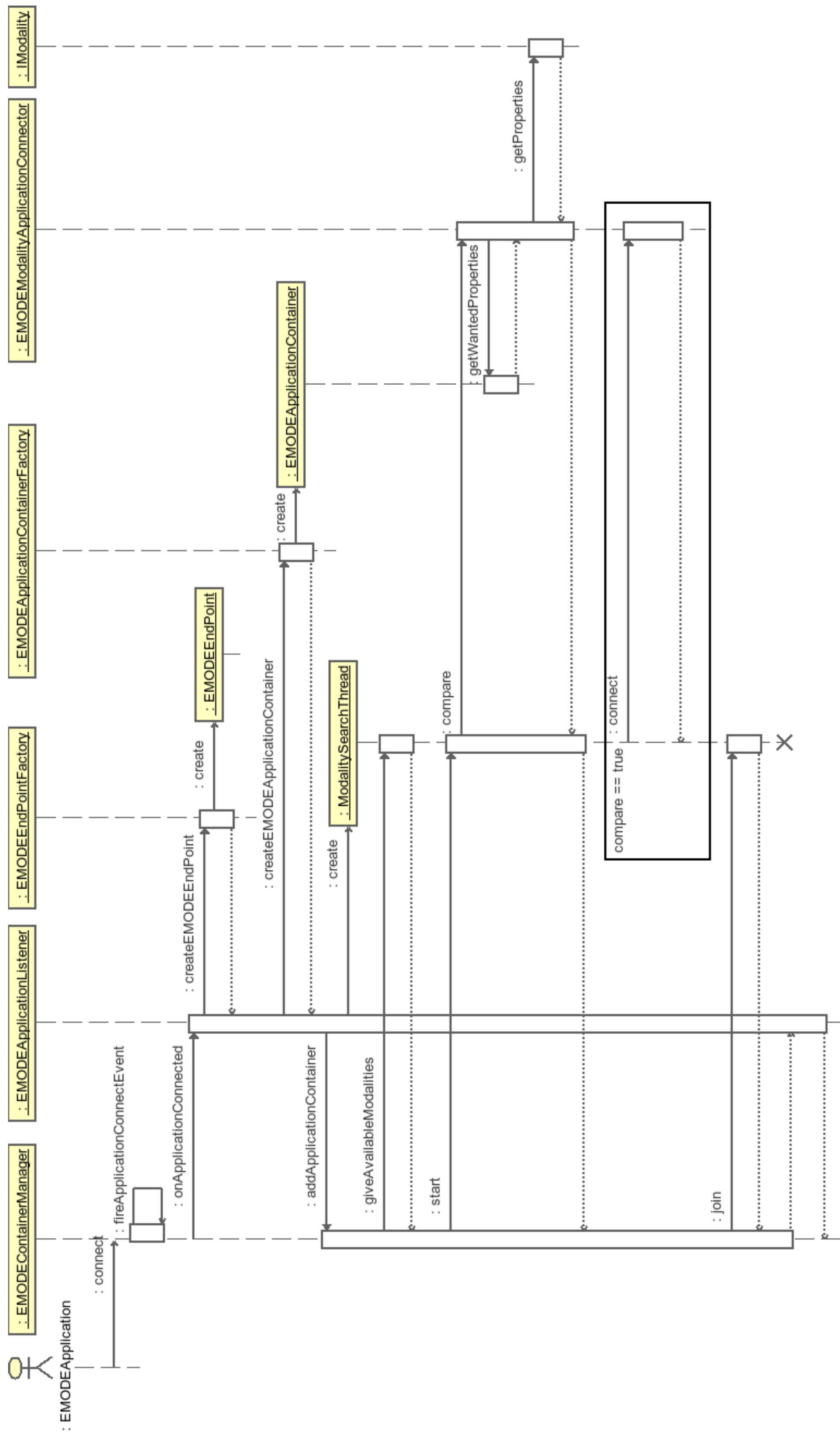
A.4.2. Aktivieren einer Modalität



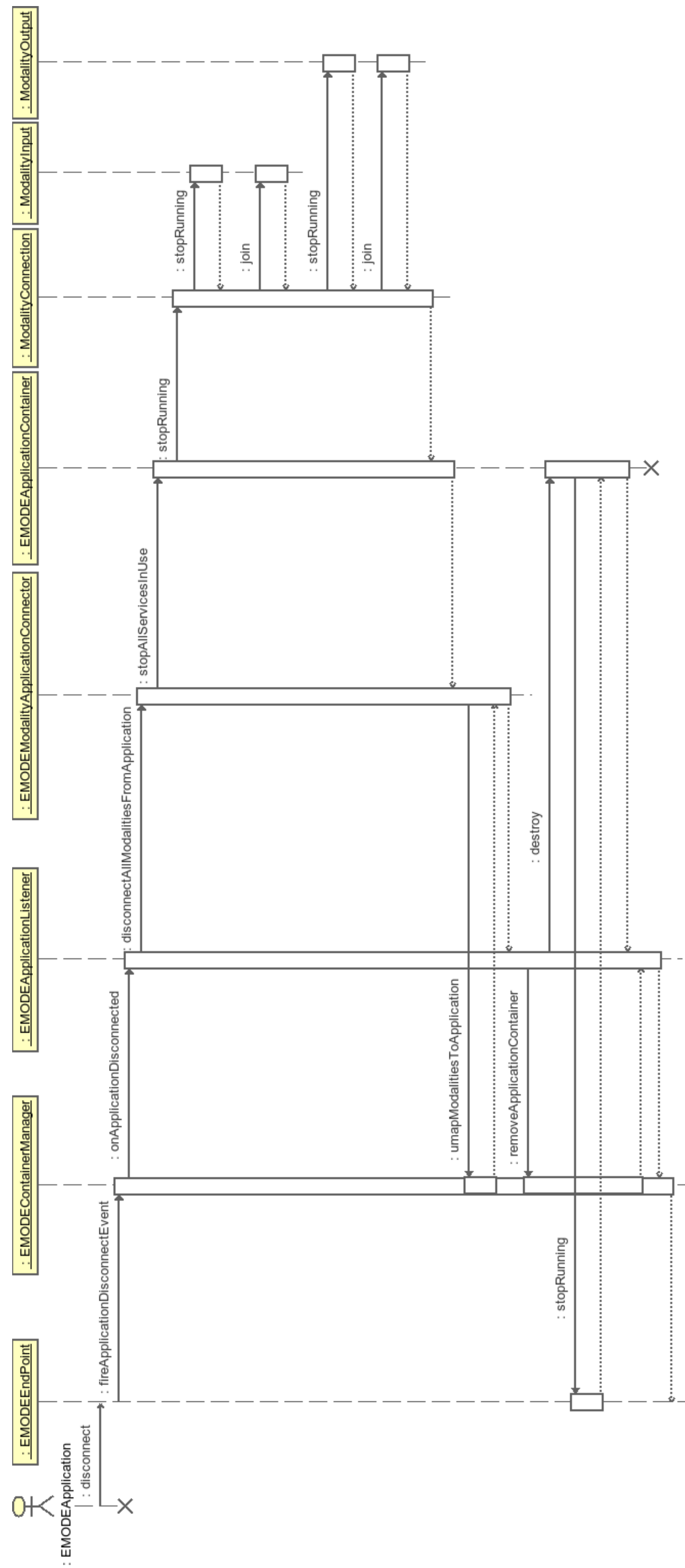
A.4.3. Deaktivieren einer Modalität



A.4.4. Starten einer EMODE Anwendung

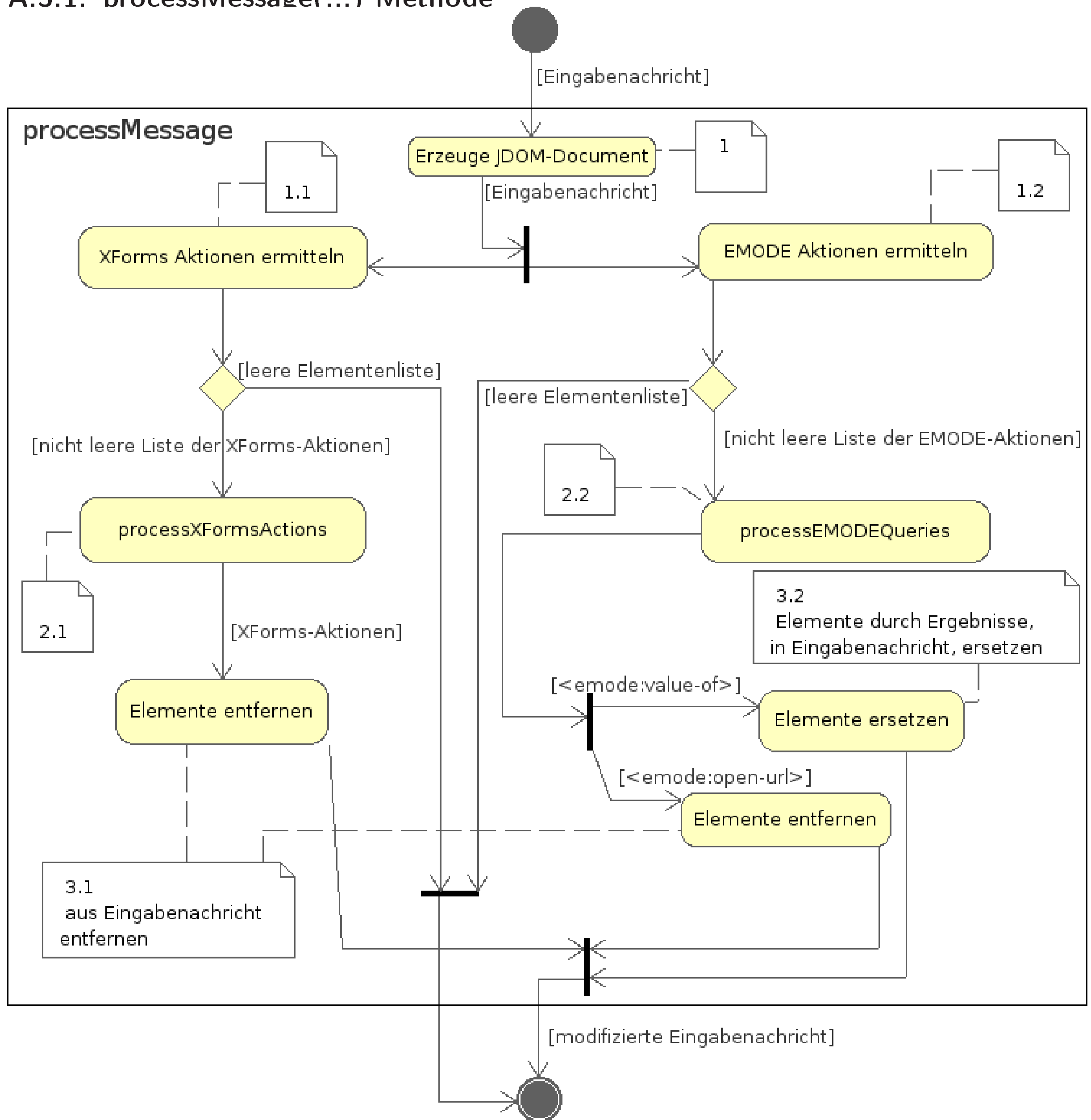


A.4.5. Beenden einer EMODE Anwendung

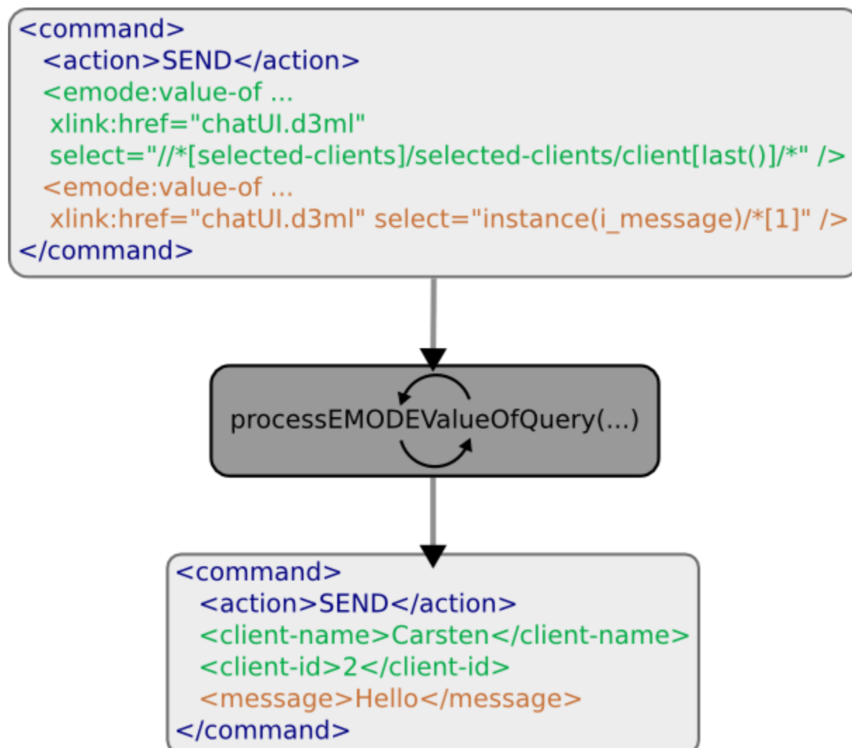


A.5. Methoden der Eingabeverarbeitung

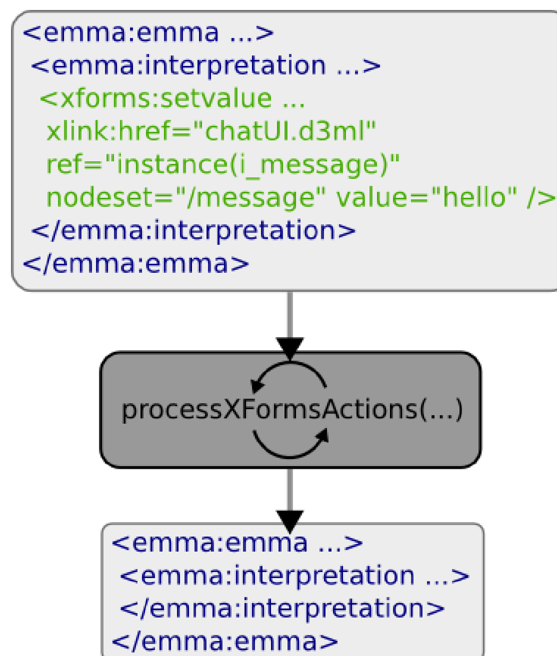
A.5.1. processMessage(...) Methode



A.5.2. Ersetzen der EMODE-Aktionen

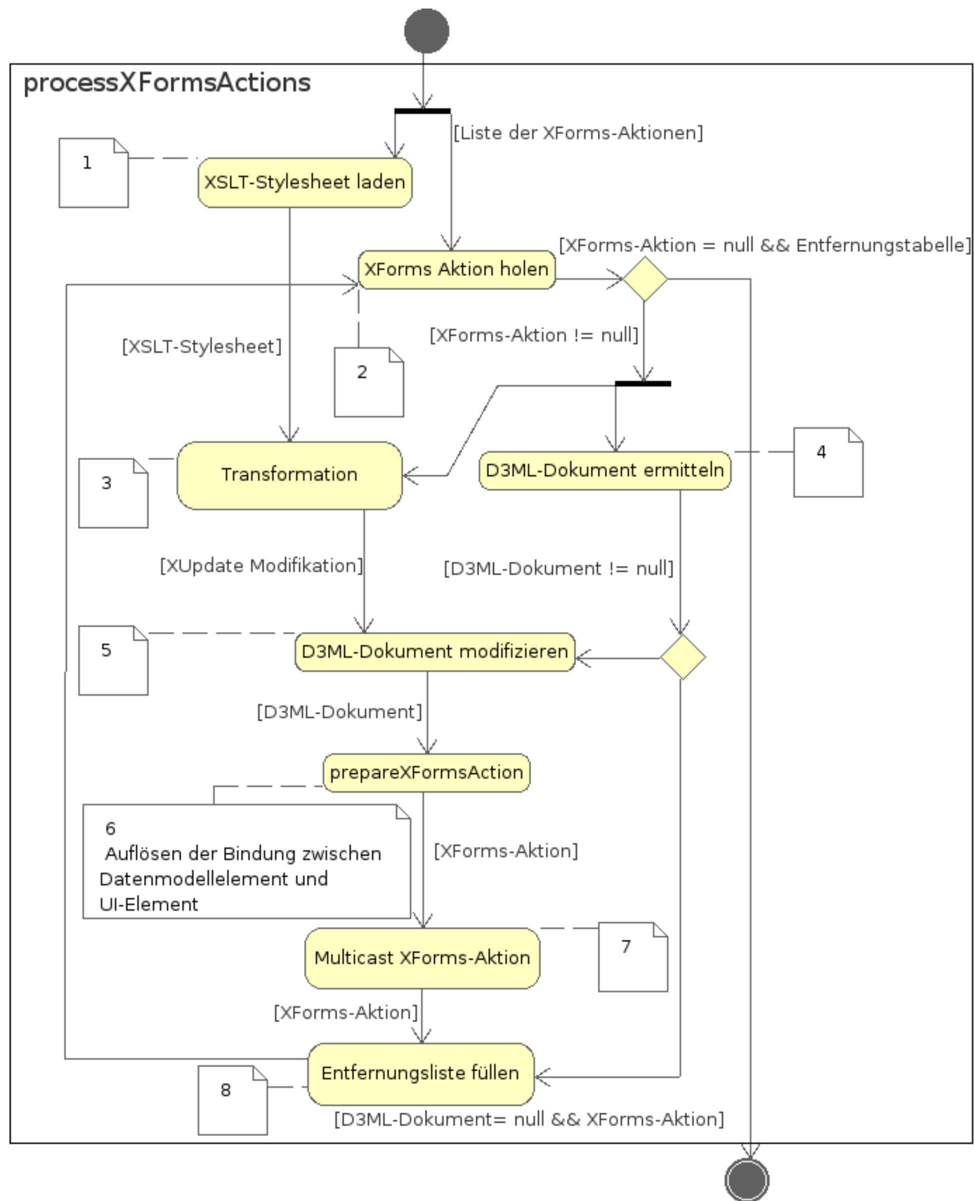


A.5.3. Entfernen der XForms-Aktionen



A.5.4. Verarbeitung der XForms-Aktionen

A.5.4.1. processXFormsActions(...) Methode



B. Quellcode

B.1. Vergleichsalgorithmus

Listing B.1: EMODEModalityApplicationConnectorImpl compare()-Methode

```
public boolean compare(IModality modality,
                        EMODEApplicationContainer application) {
  try {
    Map<String, List<String>> wanted_input = application
                              .wantedInputModalityProperties();
    Map<String, List<String>> wanted_output = application
                              .wantedOutputModalityProperties();

    // create clones to safe initial state
    Map<String, List<String>> clone_wi =
                              new Hashtable<String, List<String>>();
    Map<String, List<String>> clone_wo =
                              new Hashtable<String, List<String>>();

    deepCopy(wanted_input, clone_wi);
    deepCopy(wanted_output, clone_wo);

    Map<String, Object> mod_inputprops = modality
                                      .getInputProperties();
    Map<String, Object> mod_outputprops = modality
                                      .getOutputProperties();

    ArrayList<Object[]> checklist = new ArrayList<Object[]>();

    // check kind of modality
    if (modality.getEMMAProcessor() != null &&
        modality.getUIAccessor() == null) {
      // input modality
      checklist = checkAttributeKeys(mod_inputprops,
                                wanted_input,
                                checklist);
    }
    else if (modality.getEMMAProcessor() == null &&
            modality.getUIAccessor() != null) {
      // output modality
      checklist = checkAttributeKeys(mod_outputprops,
                                wanted_output,
                                checklist);
    }
    else if (modality.getEMMAProcessor() != null &&
            modality.getUIAccessor() != null) {
      // input & output modality
      checklist = checkAttributeKeys(mod_inputprops,
                                wanted_input,
                                checklist);
```

```
46        checklist = checkAttributeKeys(mod_outputprops,
47                                       wanted_output,
48                                       checklist);
49      }
50      else {
51        throw new Exception("EMODEModalityAppConnector: "+
52                            "modality invalid!");
53      }
54
55      if(checklist.isEmpty()){
56        // modality properties are not the wanted ones
57        return false;
58      }
59
60      // check validity -> all attribute keys have to be marked as true
61      Iterator<Object[]> valid = checklist.iterator();
62      while (valid.hasNext()) {
63        Object[] current = valid.next();
64        // is one key not ok, modality meets not wanted conditions
65        if (!((Boolean) current[1]).booleanValue()) {
66          rollback(wanted_input, clone_wi);
67          rollback(wanted_output, clone_wo);
68          return false;
69        }
70      }
71    } catch (Exception e) {
72      e.printStackTrace();
73      return false;
74    }
75    return true; // all is valid
76  }
77
78  ...
79
80  private ArrayList<Object[]> checkAttributeKeys(
81                          Map<String, Object> mod_props,
82                          Map<String, List<String>> wanted_props,
83                          ArrayList<Object[]> checklist){
84
85    Iterator<String> iter = mod_props.keySet().iterator();
86    while (iter.hasNext()) {
87      // get attribute key of modality properties
88      String cur_key = iter.next();
89
90      // list of possible attribute values
91      List<String> wanted_values = wanted_props.get(cur_key);
92
93      // test if it is in set of wanted input properties
94      if(wanted_values != null){
95
96        // current attribute value of modality;
97        // can be a String or a List of Strings
98        Object modal_value = mod_props.get(cur_key);
99
100       // if list of possible attribute values contains
101       // the attribute value of modality, mark the key as true,
102       // else false
103       Object[] entry = markKey(wanted_values, modal_value, cur_key);
```

```
104         checklist.add(entry);
105       }
106     }
107     return checklist;
108   }
109
110   private Object[] markKey(List<String> list,
111                   Object modal_value, String key) {
112
113     boolean found = false;
114     Object[] entry = new Object[2];
115
116     if(modal_value instanceof String){
117       found = list.contains((String) modal_value);
118       list.remove((String) modal_value);
119     }
120     else if(modal_value instanceof List){
121       found = list.containsAll((List) modal_value);
122       list.removeAll((List) modal_value);
123     }
124
125     // marks key as true or false
126     entry[0] = key;
127     entry[1] = new Boolean(found);
128     return entry;
129   }
```

B.2. ModalityInput

Listing B.2: `ModalityInput run()`-Methode

```
1   public void run(){
2     while(run){
3       try{
4         int read = input.read(messagebuffer);
5         if(read > 0){
6           String message = String.valueOf(messagebuffer);
7           message = message.trim();
8           message = message.substring(0, read);
9           message = chomp(message);
10
11          if(message.contains(EMODEEndPoint.XFORMS_PREFIX) ||
12             message.contains(EMODEEndPoint.EMODE_PREFIX)){
13
14            // process xforms & emode messages
15            // emode messages return result messages
16            message = messageprocessor.processMessage(message);
17          }
18
19          if(message != null && message.length() > 0){
20
21            //send message to application
22            endpoint.sendMessage(message);
23          }
24        }else{
25          // connection broken
26          run = false;
27        }
```

```
28        Thread.sleep(300);
29      }
30      catch (Exception e) {
31        ...
32      }
33    }
34  }
```

B.3. ModalityOutput

Listing B.3: `ModalityOutput run()`-Methode

```
1   public void run() {
2     while(run){
3       try{
4
5         // read application messages from endpoint
6         String endp_message = endpoint.readMessage();
7
8         // read multicast messages from other output objects
9         String multic_message = base.readMessage();
10
11        if(endp_message != null && endp_message.length() > 0){
12
13          // grab modification messages
14          if(endp_message.contains(EMODEEndPoint.MODIFICATION)){
15            endp_message = processXUpdateMessages(endp_message);
16          }
17
18          if(endp_message != null){
19            // multicast messages to other ui accessors
20            // of output modalities
21            multiCast(endp_message);
22
23            // print messages out to modality
24            output.println(endp_message);
25            output.flush();
26          }
27        }
28
29        if(multic_message != null && multic_message.length() > 0){
30          output.println(multic_message);
31          output.flush();
32        }
33        Thread.sleep(300);
34      }catch (Exception ie) {
35          run = false;
36      }
37    }
38  }
```

B.4. Aufruf der Anwendungsfunktionen

Listing B.4: Commander `call(...)`-Methode

```
1   def call(application, command, arguments)
2     if !@cmd2function.empty?() then
```

```
    begin # try block

      ...

      # fetch method name from command string
      mapentry = fetchEntry(@cmd2function, command)
      meth_sym = mapentry.function.intern()

      # get number of functions arguments
      args_num = application.method(meth_sym).arity()

      # compare wanted and given arguments with each other
      if arguments != nil &&
         scanForMethod(application, meth_sym) &&
         arguments.size() == args_num then

        # call method of application with arguments array
        application.send(meth_sym, *arguments)

      elsif (meth = application.method(meth_sym)).arity()
         != arguments.size() then
        # raise exception
       ...
      elsif !scanForMethod(application, meth_sym) then
        # raise exception
      ...
      else
       #call methode object without argument
       application.send(meth_sym)
      end
    rescue Exception => e: #catch block
      # exception handling
      ...
    end
  end
end
```

B.5. Schemadateien

B.5.1. Kommando-Funktion-Bindung

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.inf.tu-dresden.de/2007/EMODE/cmd2function"
 xmlns="http://www.inf.tu-dresden.de/2007/EMODE/cmd2function"
 elementFormDefault="qualified">

  <xs:element name="binding" type="bindingRootType"/>

  <xs:complexType name="bindingRootType">
    <xs:annotation>
      <xs:documentation>
        This element contains elements, which are describing
        a single binding of one command to one application function.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
```

```
17      <xs:element name="binding-item" minOccurs="0"
18       maxOccurs="unbounded" type="bindingItemElementType"/>
19    </xs:sequence>
20  </xs:complexType>
21
22  <xs:complexType name="bindingItemElementType">
23    <xs:sequence>
24      <xs:element name="command" minOccurs="1"
25       maxOccurs="1" type="commandElementType"/>
26      <xs:element name="function" minOccurs="1"
27       maxOccurs="1" type="functionElementType"/>
28    </xs:sequence>
29  </xs:complexType>
30
31  <xs:complexType name="commandElementType">
32    <xs:sequence>
33      <xs:element name="action" type="xs:string" minOccurs="1"
34       maxOccurs="1"/>
35      <xs:any minOccurs="0" maxOccurs="unbounded"
36       processContents="lax"/>
37    </xs:sequence>
38  </xs:complexType>
39
40  <xs:complexType name="functionElementType">
41   <xs:sequence>
42      <xs:element name="name" type="xs:string" minOccurs="1"
43       maxOccurs="1"/>
44      <xs:element name="argumentlist" minOccurs="1"
45       maxOccurs="1"/>
46    </xs:sequence>
47  </xs:complexType>
48
49 </xs:schema>
```

Literaturverzeichnis

[Boy07] Boyer, John M.: *XForms 1.1 W3C Working Draft 22 February 2007*. Technischer Bericht, W3C, http://www.w3.org/TR/xforms11/, Februar 2007.

[CDJ+04] Chou, Wu, Deborah A. Dahl, Micheal Johnston, Roberto Pieraccini und Dave Raggett: *EMMA:Extensible MultiModal Annotation*. Technischer Bericht, W3C, http://www.w3.org/TR/emma/, Dezember 2004.

[Cla99] Clark, James: *XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999*. Technischer Bericht, W3C, http://www.w3.org/TR/xslt, November 1999.

[DA05] Delgado, Ramón López-Cózar und Masahiro Araki: *Spoken, Multilingual and Multimodal Dialogue Systems, Development and Assessment*. John Wiley & Sons, Ltd, 2005.

[DMO01] DeRose, Steve, Eve Maler und David Orchard: *XML Linking Language (XLink) Version 1.0*. Technischer Bericht, W3C, http://www.w3.org/TR/xlink, Juni 2001.

[DSSR00] Dubinko, Micah, Stacy Silvester, Sebastian Schnitzenbaumer und Dave Raggett: *XForms 1.0: Data Model, W3C Working Draft*. Technischer Bericht, W3C, April 2000.

[Emb04] Emberger, Emanuel N.: *A Service-Oriented Middleware for Distributed Applications with a Multimodal User Interface*. Diplomarbeit, FH Hagenberg Software Engineering, August 2004.

[Gat06] Gatespace Telematics, www.knopflerfish.org: *Knopflerfish version 2.0.1 - Open Source OSGi*, 2006.

[HHN] Hamann, Thomas, Gerald Hübsch und René Neumerkel: *Specification of EMODE Runtime Environment*. Deliverable D3.2 – Part2, Specification of EMODE Runtime Environment.

[Hun00] Hunt, Andrew: *JSpeech Grammar Format*. W3C Working Draft, SUN Microsystems Inc., 2000.

[LM00] Laux, Andreas und Lars Martin: *XUpdate Working Draft - 2000-09-14*. Technischer Bericht, XML:DB, http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html, September 2000.

[LMF07] Lewis, Rhys, Roland Merrick und Max Froumentin: *Content Selection for Device Independence (DISelect) 1.0*. Technischer Bericht, W3C, http://www.w3.org/TR/cselection/, Juli 2007.

[LRR+03] Larson, James A., T.V. Raman, Dave Raggett, Micheal Bodell, Micheal Johnston, Sunil Kumar, Stephen Potter und Keith Waters: *W3C Multimodal Interaction Framework*. Technischer Bericht, W3C, http://www.w3.org/TR/mmi-framework/, Mai 2003.

[MLM+06] MacKenzie, C. Matthew, Ken Laskey, Francis McCabe, Peter F. Brown und Rebekah Metz: *Reference Model for Service Oriented Architecture 1.0*, Oktober 2006.

[MPR03] McCarron, Shane, Steven Pemberton und T. V. Raman: *XML Events An Events Syntax for XML.* Technischer Bericht, W3C, http://www.w3.org/TR/2003/REC-xml-events-20031014/, Oktober 2003.

[NC93] Nigay, Laurence und Joelle Coutaz: *A design space for multimodal systems: concurrent processing and data fusion.* In: *INTERCHI '93: Proceedings of the INTERCHI '93 conference on Human factors in computing systems*, Seiten 172–178, Amsterdam, The Netherlands, The Netherlands, 1993. IOS Press.

[OSG05] *About the OSGi Service Plattform*, November 2005. Technical Whitepaper, Revision 4.1.

[OSG06] OSGi Alliance: *OSGi Specification*, Juli 2006.

[SAP06] SAP: *Specification of mobile-worker UI description language final version.* Technischer Bericht, SAP, Oktober 2006.

[TFH01] Thomas, Dave, Chad Fowler und Andy Hunt: *Programming Ruby - The Pragmatic Programmer's Guide.* Addison Wesley Longman, Inc, Second Auflage, 2001.

[WLK+04] Walker, Willie, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf und Joe Woelfel: *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*, 2004.

[WLK05] Walker, Willie, Paul Lamere und Philip Kwok: *FreeTTS 1.2 - A speech synthesizer written entirely in the JavaTM programming language.* Technischer Bericht, Sun Microsystems Laboratories, http://freetts.sourceforge.net, 2005.