Konzeption und Umsetzung eines auf Statecharts basierenden Ansatzes zur kontextsensitiven Anwendungsadaption

Bernhard Ehringer

DIPLOMARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Mobile Computing

in Hagenberg

im Juni 2010

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 24. Juni 2010

Bernhard Ehringer

Inhaltsverzeichnis

Vorwort

Die vorliegende Diplomarbeit entstand im Rahmen meines Studiums an der Fachhochschule Hagenberg und beschäftigt sich mit der kontextsensitiven Anwendungsadaption im Rahmen des Studienprojekts Lomotain. Das Verfassen dieser Arbeit wäre, genauso wie das Absolvieren des gesamten Studiums, alleine nicht möglich gewesen. Deshalb möchte ich auf diesem Wege allen danken, die mich während dieser Zeit tatkräftig unterstützt haben.

Dazu zählt in erster Linie meine Familie, die mir überhaupt diese Art der Ausbildung ermöglicht hat. Auch meiner Freundin Andrea möchte ich – besonders für die ständige Motivation sowie das entgegengebrachte Verständnis – danken. Bei Herrn Dr.-Ing. Jens Krösche, der stets ein offenes Ohr für auftauchende Fragen hatte, bedanke ich mich für die tolle Zusammenarbeit während des gesamten Studiums sowie für die fachliche Betreuung dieser Arbeit. Auch Marcus Bernegger, der mit mir gemeinsam unser beider Diplomarbeitsprojekt umgesetzt hat, sowie allen weiteren Studienkollegen und Professoren möchte ich für die tolle Zeit sowie die gute, kameradschaftliche Zusammenarbeit danken.

Kurzfassung

Nahezu jeder Erwachsene besitzt heutzutage, zumindest in unseren Breiten, ein Mobiltelefon. Neben den bewährten Funktionen bieten diese mobilen Begleiter aufgrund der unglaublich kurzen Entwicklungszyklen sowie neuer Technologien bereits die Möglichkeit, Informationen der näheren Umgebung bzw. den Kontext des Benutzers (z. B. aktuelle Position, Geschwindigkeit, Temperatur, Displayauflösung) zu erfassen. So gesammelte Informationen können etwa zur Umsetzung kontextsensitiver Anwendungen benutzt werden, welche gegenüber herkömmlichen Systemen den großen Vorteil besitzen, Benutzer mit Hilfe situationsbedingter Adaptionen optimal unterstützen zu können. Dazu verwenden kontextsensitive Anwendungen diverse Konzepte, welche die Applikation hinsichtlich Visualisierung, Informationsgehalt oder Verhalten entsprechend der aktuellen Umgebung verändern und diese an die gerade herrschende Situation vollautomatisch anpassen.

Die vorliegende Diplomarbeit beschäftigt sich mit der Konzeption und Umsetzung kontextsensitiver Systeme im Rahmen des bestehenden Projekts Lomotain. Dieses hat die Entwicklung mobiler Adventure-Guides für Tourismusgebiete zum Ziel und stellt dabei besonders die Interaktion mit der Umwelt in den Vordergrund. Zur Definition der gewünschten Anwendungslogik werden Statecharts verwendet. Dieses Konzept wird von dem hier vorgestellten Ansatz erweitert und ermöglicht es künftig, sowohl Logik als auch Inhalte von mit Lomotain erstellten Anwendungen kontextsensitiv zu gestalten. Neben der genauen Erläuterung von Konzept und Umsetzung werden im Rahmen der Arbeit ebenfalls benötigte Technologien thematisiert sowie das Ergebnis eines Test-Szenarios mit bereits in der Praxis erprobten Adaptionsverfahren verglichen.

Abstract

Nowadays nearly every adult, at least in our region, owns a mobile phone. Due to incredible short development cycles as well as new technologies, these mobile companions provide, besides the established functions, the possibility to comprise information about the close surroundings respectively the actual context of the user (i. e. current position, speed, temperature, display resolution). Data gathered this way can for instance be utilised for the realisation of context-aware systems. In opposition to traditional applications, systems of this nature offer the priceless benefit to be able to support users in an optimal way by adapting themselves according to the actual context. For this purpose context-aware applications use various concepts to modify visualisation, information content or application behaviour according to the current environment and automatically adapt to the current situation.

This thesis deals with the conception and realisation of context-aware systems within the framework of the existing project Lomotain. This venture, whose goal is the development of mobile adventure guides for tourism regions, places special emphasis on the interaction with the surroundings and uses statecharts to define the desired application's logic. This concept is extended by the here introduced approach, which then enables appliances built by Lomotain to configure the application's logic as well as included contents using context sensitive aspects. In addition to the detailed explanation of concept and implementation, this thesis covers required technologies and compares results of an accomplished system test with adaption methods already approved in practice.

Kapitel 1

Einleitung

1.1 Motivation

Heutzutage gibt es bereits nahezu so viele Mobilgeräte wie Menschen auf der Erde – und kaum jemand, der ein Mobiltelefon besitzt, hat seines nicht stets dabei. Somit sind diese Geräte immer in der näheren Umgebung des Benutzers selbst – also dort, wo sie den Anwender optimal unterstützen können. Mit Hilfe einer Unmenge an Sensoren und weiteren Bauteilen entwickeln sich mobile Endgeräte derzeit zu echten Hochleistungsrechnern, welche in der Lage sind, stets Informationen – etwa über die aktuelle Umgebung bzw. die Situation – ermitteln und aufzeichnen zu können. Zu diesen dadurch gesammelten Informationen zählen beispielsweise aktuelle Position, Geschwindigkeit, Tageszeit, Umgebungstemperatur, Gemütszustand, aber auch Informationen über die verwendete Hardware wie z. B. Displaygröße, verfügbare Netzwerkverbindungen oder aktuelle Akkuladung.

Neben der simplen Aufzeichnung dieser Daten bzw. der aktuellen Situation des Benutzers (Kontextsituation) eröffnet sich vor allem die Möglichkeit, neue – kontextsensitive – Applikationen zu entwickeln. Diese bringen große Vorteile gegenüber herkömmlichen Systemen mit sich, welche üblicherweise in jeder Situation gleich reagieren bzw. die selben Informationen darstellen. Kontextsensitive Anwendungen hingegen erlauben es, sich selbst im Bezug auf Informationsgehalt, Visualisierung oder Verhalten entsprechend der aktuellen Situation anzupassen. Durch diese Integration der Applikation und deren Verhalten in die aktuelle Umgebung kann der Anwender – egal in welcher Situation er sich befindet – optimal unterstützt werden.

Die Basis von Systemen dieser Art bilden neben den erfassten Kontextinformationen meist geeignete Konzepte bzw. Mechanismen, welche die gewünschten kontextsensitiven Adaptionen durchführen. Ein Beispiel einer solchen Anpassung an den aktuellen Kontext zeigt Abbildung 1.1. Dabei wird mit Hilfe der erfassten Positionsinformation eine dargestellte Landkarte an die aktuelle, vom GPS-Modul (Global Positioning System) gelieferte, Posi-

tion verschoben und dem Benutzer somit das zeitaufwändige Navigieren zur jeweiligen Ortschaft erspart.

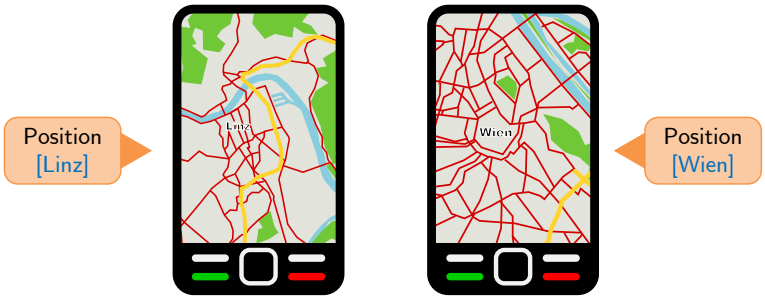


Abbildung 1.1: Kontextsensitive Beispielanwendung.

Diese einfache Beispielanwendung zeigt bereits das Potenzial solcher kontextsensitiver Anwendungen. Ein großer Vorteil – neben der Adaption selbst – ist dabei die Tatsache, dass es keine Einschränkungen bei der Erfassung von Kontextinformationen bzw. Kontextparametern gibt – Erweiterungen in alle Richtungen sind denkbar. Außerdem ist es möglich, mit Hilfe diverser Kontextparameter komplexere Informationen – ganz ohne hardwareseitige Sensoren – zu berechnen oder neben dem aktuellen Kontext auch Informationen aus der Vergangenheit zur optimalen Anwendungsadaption zu nutzen. Des Weiteren können extern abgerufene Daten (z. B. Wetterbericht) helfen, noch mehr Informationen über die aktuelle Umgebung zu sammeln und dadurch die Applikation noch gezielter an die gegenwärtige Situation anzupassen.

1.2 Herausforderungen

Die Thematik der Kontextsensitivität bietet eine Menge interessanter Herausforderungen. Dazu zählen etwa das Identifizieren geeigneter Kontextparameter sowie das Erfassen dieser benötigten Informationen, welche die Grundlage jeder kontextsensitiven Applikation darstellen. Aus dieser Akquisition der erforderlichen Daten ergeben sich weitere Herausforderungen – etwa das Bereitstellen eines generischen Frameworks zur Erfassung unterschiedlichster Sensorwerte bzw. Informationen aus weiteren Datenquellen (z. B. Webservices). Zur optimalen Darstellung und Verarbeitung dieser gesammelten Kontextparameter innerhalb der Anwendung ist es notwendig, ein adäquates Datenmodell zu entwerfen bzw. zu finden. Um im späteren Verlauf Adaptionen durchführen zu können, müssen außerdem Verfahren entwickelt werden, um die erfassten Kontextinformationen an die Anwendung bzw. die betreffenden Module weiterzureichen. Eine weitere Herausforderung – die Umsetzung eines Update-Mechanismus zur Aktualisierung der integrierten Kontextinformationen – ergibt sich aus dem Wunsch nach stets aktuell ge-

haltenen Daten bzw. der Möglichkeit, auch während der Anwendungslaufzeit auf Kontextänderungen reagieren zu können.

Eine Herausforderung, welche aus dem typischen Verhalten – der situationsbedingten Anwendungsadaption – kontextsensitiver Applikationen resultiert, ist das Finden eines geeigneten Adaptionsverfahrens bzw. das Extrahieren und Untersuchen bestehender Ansätze, welche von existierenden Anwendungen verwendet werden. Auch das verwendete Adaptionsverfahren selbst bringt einige Herausforderungen mit sich – beim Einsatz von Regel-Mechanismen ist es etwa notwendig, eine geeignete Syntax zu erstellen, mit welcher die gewünschten Bedingungen aufgestellt werden können. Im Rahmen der Anwendungsadaption ist es des Weiteren notwendig, die zur jeweiligen Applikation passende Adaptionsart (z. B. Visualisierung, Verhalten, Informationsgehalt) zu eruieren und eine geeignete Technik zur Umsetzung zu entwickeln.

Soll ein existierendes System kontextsensitiv gestaltet werden, ergeben sich wiederum einige Herausforderungen. Hierzu zählt etwa das Entwickeln geeigneter Methoden, um mit den von dieser Applikation genutzten Technologien kontextsensitive Aspekte in die Anwendung integrieren zu können.

1.3 Zielsetzung

Ziel der Arbeit ist es, in einem ersten Schritt zu prüfen, welche vorhandenen kontextsensitiven Systeme es gibt und welche Konzepte dabei verwendet werden, um die Anwendung – bezogen auf den teilweise oft wechselnden Kontext – zu adaptieren.

Die Erstellung eines Konzepts zur Umsetzung kontextsensitiver Anwendungen auf Basis von Statecharts definiert das Hauptziel dieser Diplomarbeit. Dies ergibt sich aus der Tatsache, dass bei vielen bestehenden Anwendungen die Adaptionsregeln entweder direkt in den Source-Code integriert sind oder durch komplexe Regel-Mechanismen definiert werden und somit relativ schwer zugänglich, darzustellen und zu verstehen sind. Im Rahmen dieser Arbeit soll geprüft werden, ob das Konzept der Statecharts (bisher vorwiegend verwendet, um komplexe Szenarien wie eine Anwendungslogik vereinfacht und abstrakt darzustellen) diesen Nachteilen entgegenwirken kann und die verständliche Visualisierung kontextsensitiver Aspekte und somit die erfolgreiche Umsetzung situationsabhängiger Anwendungen ermöglicht.

Um dieses Ziel zu erreichen, sollen diverse Möglichkeiten bzw. Variationen verglichen und das geeignetste Verfahren zur Integration von Kontextsensitivität in Statecharts gefunden werden. Ebenfalls ist es notwendig, einige Kontextparameter zu identifizieren, welche für diese Arbeit sowie für das zugehörige Projekt geeignet erscheinen. Darauf aufbauend soll eine eigene Regelsyntax für jene Bedingungen definiert werden, welche das Anpassen der Applikation an die Kontextsituation steuern sollen.

Neben dem Erstellen eines Konzepts, welches als Grundlage für die Umsetzung dient, ist auch die Implementierung selbst ein wichtiges Ziel und zentraler Bestandteil der Arbeit. Dabei soll das an der Fachhochschule Hagenberg entwickelte Projekt Lomotain – dieses verwendet bereits Statecharts zur Definition der Spiellogik – um einige Aspekte (u. a. Kontextsensitivität) erweitert werden.

Das letzte große Ziel der Arbeit ist ein Vergleich des hier vorgestellten Verfahrens mit bestehenden, in der Praxis bereits erprobten Adaptionskonzepten. Dabei soll die Entwicklung eines Test-Szenarios helfen, die Vor- und Nachteile des auf Statecharts basierenden Ansatzes zu identifizieren sowie diesen auf Benutzerfreundlichkeit und Praxistauglichkeit zu prüfen.

1.4 Gliederung

Im folgenden Kapitel 2 werden einerseits Grundlagen zum Thema Kontext und Kontextsensitivität behandelt. Andererseits beinhaltet das Kapitel Abschnitte, in denen bereits existierende kontextbezogene Anwendungen vorgestellt und im Hinblick auf verwendete Adaptionskonzepte bewertet werden.

Kapitel 3 stellt die für die Umsetzung der Arbeit verwendeten Technologien vor – darunter Statecharts, XML und JavaCC. Ebenfalls wird auf die Eclipse-Platform[1] eingegangen, da ein großer Teilbereich des Lomotain-Projekts, das Admin-Tool zum Erstellen von Spielszenarien, darauf basiert.

Dem mittels Statecharts umgesetzten kontextsensitiven Ansatz sowie der Eingliederung dessen in das Gesamtprojekt Lomotain widmet sich Kapitel 4. Dabei werden sowohl diverse Lösungsansätze näher thematisiert als auch Einblick in Architektur und Umsetzung aller beteiligten Teilbereiche gegeben. Ein vorgestelltes Test-Szenario zeigt außerdem – im Vergleich mit bisherigen Adaptionskonzepten – die Vorteile des im Rahmen dieser Arbeit entwickelten Verfahrens.

Den Schluss bildet Kapitel 5 mit einer kurzen Zusammenfassung und Bewertung der Arbeit.

[1] http://www.eclipse.org

Kapitel 2

Stand der Technik

2.1 Grundlagen

Dieses Kapitel beschäftigt sich mit der Aufarbeitung von Begrifflichkeiten rund um das Thema Kontextsensitivität. Des Weiteren werden bestehende Konzepte zu Kontext-Modellierung und -Adaption vorgestellt und hinsichtlich wichtig erscheinender Kriterien bewertet.

2.1.1 Kontext

Der Begriff *Kontext* ist in der Literatur nur vage definiert und wird von vielen Menschen unterschiedlich interpretiert. Da alles und jedes in einem gewissen Kontext abläuft, ist es relativ schwierig, eine allgemeine Definition zu finden. Meist wird der Kontext mit dem Umfeld, in dem sich ein Objekt – etwa eine Person – befindet, assoziiert. Eine oft zitierte Definition des Begriffs wurde von Schilit [49] aufgestellt:

> Three important aspects of context are: where you are, who you are with, and what resources are nearby.

Dies besagt, dass nicht nur der Ort eine Rolle spielt, sondern auch weitere nicht immer gleich bleibende Aspekte wie etwa der Lärmpegel, die Lichtverhältnisse oder die aktuelle soziale Situation den Kontext beeinflussen.

Des Weiteren wird der Begriff *Kontext* von Schilit in drei verschiedene Kategorien unterteilt, welche von Chen und Kotz [11] – fokussierend auf den Anwendungsbereich Mobile Computing – um zwei weitere Bereiche (time context, context history) ergänzt werden:

- **Computing context:** Informationen über zur Verfügung stehende Computer (z. B. Displayauflösung)
- **User context:** Informationen über den Benutzer (z. B. soziale Situation)

- **Physical context:** Physische Bedingungen (z. B. Vibration, Temperatur, Wetter)
- **Time context:** Zeitliche Informationen zur aktuellen Situation (z. B. Wochentag)
- **Context history:** Kontextinformationen, die frühere Zeitpunkte beschreiben

Außerdem teilen sie den Kontextbegriff im Rahmen von Mobile Computing in zwei Arten. Einerseits den aktiven Kontext, dessen Aspekte wichtig für die mobile Anwendung sind und diese direkt beeinflussen. Andererseits gibt es den passiven Kontext, welcher alle Aspekte des Umfelds inkludiert, die zwar wichtig sind, aber die Anwendung nicht direkt beeinflussen.

Als zweite betrachtete Definition von Kontext, welche auch im Rahmen dieser Arbeit verwendet wird, dient die von Dey und Abowd [1]:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Dabei wird Kontext als jegliche Art von Information gesehen, welche die Situation einer Entität beschreibt. Eine Entität stellt in diesem Fall eine für die Interaktion zwischen Benutzer und Anwendung wichtige Person, einen Ort oder ein Objekt dar. Neben der Begriffsdefinition werden von Dey und Abowd außerdem einige Kontextparameter festgelegt, welche wichtiger als andere erscheinen. Dieser sogenannte Primär-Kontext besteht aus:

- **Lokalität:** Informationen über den Ort
- **Identität:** Informationen über die im Mittelpunkt stehende Entität
- **Aktivität:** Informationen über die Aktionen, welche in der Situation passieren
- **Zeit:** Informationen über den aktuellen Zeitpunkt

Als zweite Schicht unter diesen vier Kategorien gibt es den Sekundär-Kontext, welcher aus Attributen besteht und über den Primär-Kontext identifiziert werden kann. So ist etwa die Adresse einer Identität ein Element des Sekundär-Kontexts, welches über die Identität selbst (etwa den Namen in einem Telefonbuch) erreicht werden kann.

Fuchs [23] beschreibt eine weitere Möglichkeit, Kontextdaten in mehrere Kategorien (Low- und High-Level-Kontextinformationen) zu gruppieren. Die Gruppe der Low-Level-Kontextinformationen stellt dabei Messwerte von Hardware-Sensoren dar und kann in mehrere Arten unterteilt werden:

- **Position:** Die Position stellt in vielen kontextsensitiven Systemen eine zentrale Rolle dar. Die nötigen Informationen können dabei unter anderem mit Hilfe einer satellitengestützten Ortung (z. B. GPS – Global Positioning System) oder einer zellbasierten Lokalisierung (z. B. WLAN – Wireless Local Area Network) erfasst werden.
- **Zeit:** Sensoren zur Messung bzw. Erfassung der aktuellen Zeit sind in nahezu jeglichen elektronischen Geräten vorhanden – somit auch auf für kontextsensitive Systeme meist verwendeten Mobilgeräten.
- **Optische Informationen:** In diesem Bereich werden etwa Helligkeitssensoren eingesetzt, um zum Beispiel zwischen Tag und Nacht zu unterscheiden.
- **Akustische Informationen:** Um bspw. auf Geräusche der Umgebung reagieren zu können, werden zumeist Mikrofone zur Messung der Akustik verwendet.
- **Sonstige:** Weitere für spezielle Anwendungsfälle interessante Informationen liefern etwa Pulsoxymeter (Ermittlung der Sauerstoffsättigung), Beschleunigungssensoren oder berührungssensitive Elemente.

Unter High-Level-Kontextinformationen beschreibt Fuchs Daten, welche nicht direkt von einem Sensor stammen sondern durch die Verarbeitung der Low-Level-Informationen entstehen. Beispiele dafür sind die aktuelle soziale Situation des Benutzers, die Gemütslage oder die gerade ausgeführte Tätigkeit. Da diese High-Level-Kontextinformationen jedoch mit Hilfe umfangreicher Regeln und/oder künstlicher Intelligenz erstellt werden, ist die Verarbeitung solcher Daten relativ komplex und teils mit großem Aufwand verbunden.

Da es in der Aufbereitung von Kontextinformationen diverse Fehlerquellen gibt, ist es ebenfalls notwendig, Aussagen über die Qualität der Daten (QoC – Quality of Context) treffen zu können. Dabei wird von Buchholz die Qualität von Kontext als Information bezeichnet, welche die Qualität der kontextbezogenen Daten beschreibt [9]. Jedoch ist hierbei zu beachten, dass sich die Qualität des Kontexts nur auf die Information selbst bezieht, nicht jedoch auf etwaige Komponenten, welche die Daten liefern.

2.1.2 Kontextsensitivität

Der Kontext – also die Informationen über eine bestimmte Situation – kann nun herangezogen werden, um eine Anwendung oder ein System zu beeinflussen. Diese Beeinflussung bzw. Anpassung stellt die Kontextsensitivität dar und wird von Schilit [49] im Bezug auf Applikationen folgendermaßen definiert:

> Kontextsensitive Anwendungen sind Systeme, die sich an den Einsatzort, die in der Umgebung befindlichen Personen und Ressourcen sowie die kontinuierliche Veränderung des Kontexts anpassen.

Des Weiteren definiert Schilit Kategorien kontextsensitiver Anwendungen, welche sich darin unterscheiden, entweder Informationen zu liefern oder Aktionen auszuführen, wobei dies entweder manuell durch den Benutzer oder automatisch geschieht.

Dey und Abowd [1] hingegen definieren kontextsensitive Systeme als Anwendungen, welche bestimmte Kontextinformationen nutzen, um dem Anwender situationsbedingt Informationen bzw. Dienste bereitzustellen. Dabei werden des Weiteren drei wichtige Funktionen kontextsensitiver Anwendungen aufgestellt, welche nahezu alle Systeme dieser Art bieten:

- Darstellung von Informationen bzw. Diensten
- Automatische Ausführung von Diensten
- Zuordnung von Kontextinformationen zu bestimmten Situationen zur späteren Verwendung

Außerdem wird Kontextsensitivität von Chen und Kotz [11] in zwei verschiedene Arten unterteilt – je nachdem wie die Kontextinformationen genutzt werden. Einerseits gibt es dabei die aktive Kontextsensitivität, welche dazu führt, dass sich Anwendungen automatisch an die ermittelte Kontextsituation anpassen. Die zweite Art – die passive Kontextsensitivität – dient hingegen nur dazu, die erforschten Kontextinformationen dem Anwender zu präsentieren bzw. diese persistent für einen späteren Zugriff abzulegen.

Ein Beispiel einer aktiven Kontextsensitivität zeigt Abbildung 2.1.

Abbildung 2.1: Aktive Kontextsensitivität.

Dabei ist ersichtlich, dass die Anwendung basierend auf einen eingelesenen Kontextparameter adaptiert wird – hier in Form des Begrüßungstextes je nach Tageszeit.

2.1.3 Struktur kontextsensitiver Anwendungen

Kontextsensitive Systeme können auf viele verschiedene Arten entworfen und umgesetzt werden – einige Aspekte sind jedoch in jedem Ansatz unbedingt notwendig, um die gewünschte kontextsensitive Funktionalität erreichen zu können. Abbildung 2.2 zeigt schematisch die zumindest benötigten Anwendungsmodule, welche im Folgenden genauer erläutert werden.

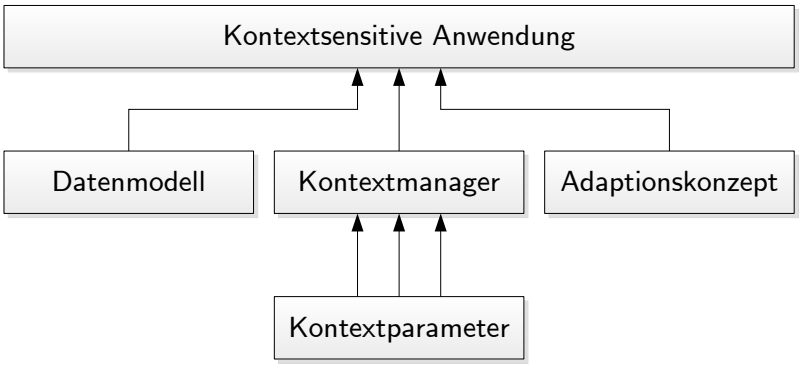


Abbildung 2.2: Struktur einer kontextsensitiven Anwendung.

Kontext-Modellierung

Dem System liegt üblicherweise ein Datenmodell zugrunde, welches die Informationen der aktuellen Situation in einer geordneten und einheitlichen Struktur ablegt. Der aktuelle Wert und der Typ der Kontextinformation sind dabei wichtige Komponenten. Abschnitt 2.2 beschreibt einige Datenmodelle, welche in der Praxis Verwendung finden.

Kontext-Management

Die Informationen, welche das zuvor entworfene Datenmodell speisen sollen, werden meist von einem Management-Modul erfasst. Dieses ist neben dem Erfassen von Low- und High-Level-Kontextparametern auch für den Austausch von Daten mit anderen Systemen verantwortlich. Dies bedeutet etwa die Kontaktaufnahme mit anderen Computern, welche mit Sensoren ausgestattet sind, die das eigene System nicht zur Verfügung hat. Nach dem Miteinbeziehen der räumlichen Differenz der beiden Systeme können beliebige Sensorwerte zwischen den Anwendungen ausgetauscht werden. Die so erhaltenen Daten können genauso wie durch eigene Sensoren erfasste Low-Level-Kontextinformationen behandelt und verarbeitet werden.

Sämtliche Kontextinformationen bilden gemeinsam eine Kontextsituation. Diese beinhaltet alle für einen diskreten Zeitpunkt vorhandenen Informationen und wird vom Kontext-Manager allen weiteren Modulen zur Ver-

fügung gestellt. Ist eine zeitliche Veränderung der Informationen für die Anwendung relevant, so ist es auch möglich, die Kontextdaten zu diskreten Zeitpunkten persistent abzulegen und eine History-Funktion bereitzustellen.

Adaptionskonzept

Da die kontextbezogenen Informationen alleine keinen Einfluss auf die Applikation selbst haben, wird ein Konzept zur Anwendungsadaption benötigt. Dabei ist der Einsatz von einfachen `if`- bzw. `case`-Statements, welche direkt in den Source-Code eingebettet sind, ebenso möglich wie die Verwendung umfangreicher und flexibler Regelmechanismen. Zwingend vorgesehen, um eine Adaption durchführen zu können, sind jedoch die Definition einer Regel bzw. einer zugehörigen Situation sowie einer passenden Aktion, welche die Adaption selbst durchführt. Abschnitt 2.3 stellt einige bestehende Anwendungen inklusive verwendeter Adaptionskonzepte genauer vor.

2.2 Kontext-Modellierung

Um Kontextinformationen in einer Anwendung darstellen und zwischen verschiedenen Systemen austauschen zu können, ist es notwendig, diese in einer einheitlichen Struktur abzulegen. Solche Strukturen enthalten i. d. R. den Typ der Kontextinformation, einen aktuellen Wert sowie Zusatzinformationen (Metadaten). Diese Metainformationen stellen strukturierte Daten über ein Objekt dar, mit Hilfe derer Funktionen ausgeführt werden können – etwa das Suchen von Personen mittels der Metainformation *Vorname* [24].

Laut Strang und Linnhoff-Popien [57] gibt es einige weit verbreitete Datenmodelle, welche in den folgenden Abschnitten genauer vorgestellt werden.

2.2.1 Key-Value-Modelle

Dieses Datenmodell stellt ein rudimentäres und sehr einfaches Konzept dar, wie Kontextinformationen in Datenstrukturen abgebildet werden können. Dabei besteht jede Dateneinheit aus einem Key (Name der Kontextinformation) und einem Wert. Diese Art der Repräsentation der Informationen wurde zu Beginn der Entwicklung kontextsensitiver Systeme eingesetzt und findet bis heute Verwendung [47]. Ein Nachteil dieser Darstellungsform ist jedoch, dass diese für Systeme, welche neben dem aktuellen Wert der Kontexteigenschaft weitere Zusatzinformationen (Metadaten) benötigen, keine geeignete Basis bildet. Ein Beispiel des Key-Value-Modells zeigt Abbildung 2.3.

2.2.2 Markup-Scheme-Modelle

Im Unterschied zu dem zuvor beschriebenen Modell sind Markup-Scheme-Modelle hierarchisch aufgebaut und beinhalten Tags mit Attributen und

```
<Position>48.36856;14.51409</Position>
<Date>2010-11-14</Date>
```

Abbildung 2.3: Key-Value-Modell.

Werten. Dies bedeutet, dass Kontextelemente somit durch mehrere Werte beschrieben werden können. Beispiele für diese Art der Darstellung sind sogenannte Profile – etwa das UAProf (User Agent Profile) oder das CC/PP (Composite Capabilities/Preferences Profile). Zu diesen Profilen, welche oft mittels XML umgesetzt werden, gibt es außerdem einige Erweiterungen. Diese bieten mehr Attribute und Möglichkeiten, Kontextdaten darzustellen. Abbildung 2.4 zeigt ein Beispiel einer Modellierung von Kontextdaten auf diese Weise.

```
<Hardware>
  <Screen size="800;600" colors="16mio"/>
</Hardware>
<Software>
  <Platform os="Windows CE" memory="128mb"/>
</Software>
```

Abbildung 2.4: Markup-Scheme-Modell.

Aufgrund der Tatsache, dass diese Modelle jedoch spezifisch für verschiedene Bereiche sowie limitiert in der Unterstützung von Kontextaspekten sind [36], ist es nicht in jedem System möglich, dieses Datenmodell zu verwenden.

2.2.3 Grafische Modelle

Mit Hilfe der UML (Unified Modeling Language) ist es möglich, Kontextaspekte grafisch abzubilden. Vor allem verschiedene Arten von Diagrammen aus der UML können dabei nützlich sein, um die Verständlichkeit der dargestellten Konzepte zu erhöhen. Bauer [4] geht in seiner Arbeit besonders auf das Modellieren von Kontextinformationen auf dem Gebiet der Luftfahrt ein. Eine laut Bauer mögliche Art, um Kontext in UML-Diagramme einzubinden, ist es, eigene Kontextelemente zu definieren und diese mit dem gewünschten Objekt unter Vewendung von Verbindungen zu verknüpfen. Somit soll es möglich sein, dass das Kontextobjekt je nach aktueller Kontextsituation Informationen an das eigentliche Objekt weiterleitet. Ein Beispiel hierzu zeigt Abbildung 2.5, wobei auf das Objekt *Flugzeug* zwei Kontextaspekte einwirken – Turbulenzen und luftfahrtspezifische Konflikte (MTC – Medium Term Conflict).

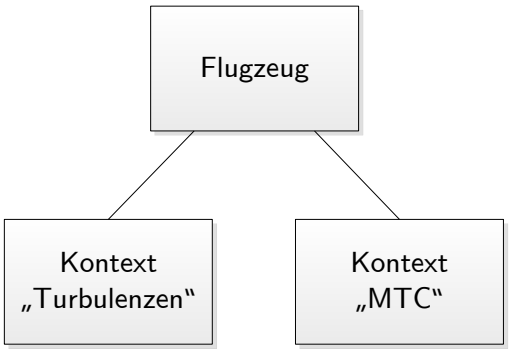


Abbildung 2.5: Grafisches Modell (nach [4]).

Neben der UML existieren weitere Möglichkeiten, um Kontext grafisch darzustellen. Hierzu zählt bspw. ORM (Object-Role Modeling), das von Henricksen [28] näher erläutert wird.

2.2.4 Objektorientierte Modelle

Durch die Verwendung von objektorientierten Techniken zur Darstellung von Kontextdaten können alle damit assoziierten Konzepte, zum Beispiel Kapselung, Vererbung und Wiederverwendbarkeit, genutzt werden [2]. Damit ist es möglich, für verschiedene Kontexttypen unterschiedliche Objekte zu erstellen – angepasst an deren jeweilige Eigenschaften. Um die Interoperabilität zwischen mehreren Modulen zu gewährleisten, ist es nötig, die Schnittstellen der Kontextobjekte mittels Interfaces festzulegen.

Ein Beispiel eines solchen objektorientierten Ansatzes zur Verarbeitung von Kontextinformationen stellt das Framework Hydrogen [29] dar. Es besteht unter anderem aus Kontextobjekten verschiedener Typen – dargestellt in Abbildung 2.6.

Dabei ist ersichtlich, dass es in oberster Ebene ein Objekt vom Typ *Context* gibt, welches den gesamten Kontext darstellt und aus Unterelementen besteht. Diese einzelnen Kontextobjekte wiederum repräsentieren zum Beispiel Informationen über einen Zeitpunkt, einen Ort oder einen Benutzer. Folgende fünf Typen von Kontextparametern werden derzeit vom System unterstützt (durch Ableiten von *ContextObject* beliebig erweiterbar):

- **Time:** Die aktuelle Uhrzeit
- **Location:** Die aktuelle Position (GPS-Koordinaten)
- **Device:** Typ und ID des verwendeten Endgeräts
- **User:** Der Name des aktuellen Benutzers
- **Network:** Verfügbare Netzwerkschnittstellen

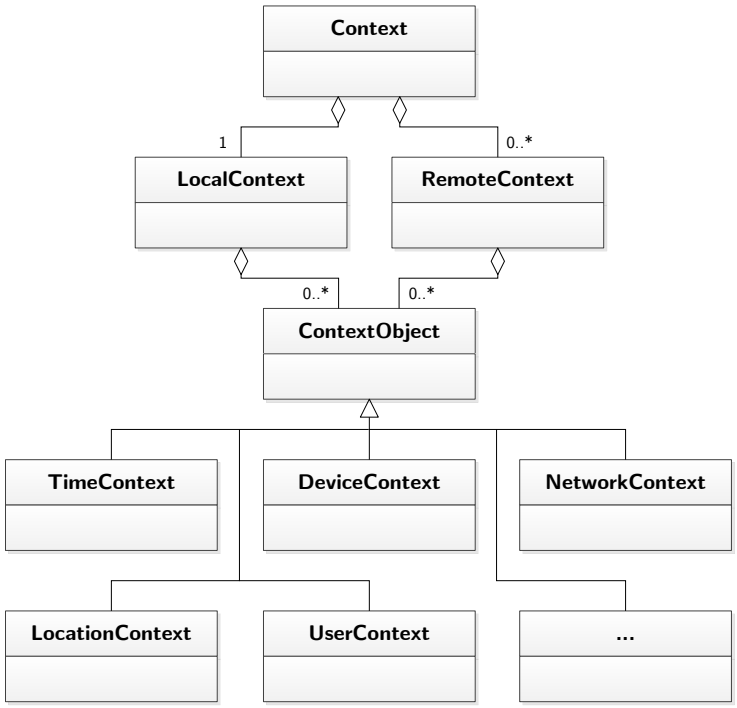


Abbildung 2.6: Objektorientiertes Modell (nach [29]).

Ein Problem der objektorientierten Modelle ist es, dass die Objekte in verschiedenen Systemen wahrscheinlich unterschiedlich aussehen und somit auch die Schnittstellen anders definiert sind [36]. Deshalb ist es schwierig, unterschiedliche Systeme miteinander zu verbinden und Kontextinformationen auszutauschen.

2.2.5 Logikbasierte Modelle

Modelle, welche nach diesem Ansatz erstellt werden, besitzen einen hohen Grad an Formalität, da dabei Kontext oftmals mit Hilfe von Fakten, Regeln und formalen Ausdrücken definiert wird [2]. Aufgrund dieser Tatsache ist es relativ einfach möglich, neue Elemente (High-Level-Kontextinformationen) zu erstellen, da dafür klare Regeln spezifiziert werden können. In ein logikbasiertes Modell werden im Normalfall neue Fakten eingespielt, alte gelöscht bzw. Daten aktualisiert.

Eines der ersten logikbasierten Modelle, das des Weiteren Vererbung ermöglicht, entwickelten McCarthy und Buvac [35]. Dabei werden Kontextinformationen als abstrakte mathematische Elemente mitsamt bestimmten Eigenschaften betrachtet. Als Basis-Relation wurde

$$ist(c,\ p)$$

definiert. Diese Formel beschreibt die Behauptung, dass die Aussage p im Kontext c wahr ist (*is true*). Damit könnte folgende Beispiel-Behauptung aufgestellt und geprüft werden:

ist(context-of(„FH OÖ“), „Mobile Computing ist ein Masterstudiengang“)

Neben diesen Behauptungen können auch Werte, welche von der aktuellen Kontextsituation abhängen, abgefragt werden:

value(context-of(„FH OÖ“), „Anzahl an Studenten“) > 500

2.2.6 Ontologiebasierte Modelle

Als Ontologien werden im Bereich der Informatik strukturierte Konzepte bezeichnet, welche in Graphen abgebildet sind und mit Hilfe von Relationen (semantische Relation oder Verbindung bzw. Vererbung) verknüpft werden [36]. Ontologien werden hauptsächlich dazu verwendet, um das Wissen des behandelten Domänenbereichs – auch für Computer verstehbar – zu modellieren. Einsatz finden Ontologien etwa beim Austausch von Wissen, da die gewünschten Konzepte mitsamt Attributen gut strukturiert werden können. Da das Aussehen und die Struktur einer Ontologie immer vom Ersteller abhängig ist, gibt es für spezielle Wissensbereiche viele unterschiedliche, jedoch gültige Ontologien.

Ein Beispiel einer einfachen Ontologie illustriert Abbildung 2.7.

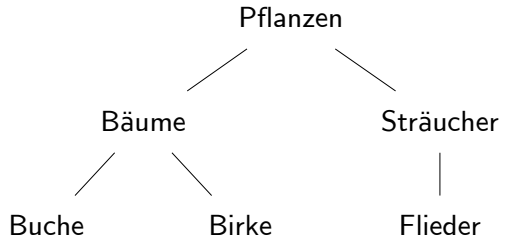


Abbildung 2.7: Ontologie zur Vegetation.

Als einer der ersten Ansätze, Kontext mittels Ontologien abzubilden, gilt das Konzept von Ötztürk und Aamodt [41]. Dabei geht es um die Analyse psychologischer Studien in Kombination mit kontextsensitiven Informationen. Auch das W3C (World Wide Web Consortium) bietet eine Möglichkeit Ontologien zu erstellen. Mit Hilfe der OWL (Web Ontology Language) können Ontologien basierend auf dem RDF (Resource Description Framework) verfasst werden.

Strang und Linnhoff-Popien kamen in ihrer Evaluierung der verschiedenen Datenmodelle zum Schluss, dass Ontologien die meisten Möglichkeiten

bieten und die von ihnen aufgestellten Anforderungen an solche Konzepte am besten erfüllen. Korpipää [31] beschreibt diesbezüglich einige Anforderungen an Ontologie-Konzepte, welche im Zusammenhang mit Kontextsensitivität verwendet werden:

- **Einfach:** Um die Arbeit der Entwickler zu erleichtern, sollen die Ausdrücke und Beziehungen so einfach wie möglich gehalten werden.
- **Flexibel und erweiterbar:** Um spätere Erweiterungen zu ermöglichen, soll die Ontologie flexibel im Bezug auf neue Kontextinformationen und Beziehungen sein.
- **Generisch:** Die Ontologie soll verschiedene Typen von Kontextobjekten unterstützen.
- **Aussagekräftig:** Um die Kontextinformationen so genau wie möglich erfassen zu können, soll die Ontologie so viele Elemente wie möglich in hohem Detailgehalt beinhalten können.

2.2.7 ContextUML

Als eine weitere Möglichkeit, Kontext in Anwendungen zu modellieren, gilt der Ansatz von Sheng und Benatallah [53]. Dieser basiert auf UML und stellt ein Metamodell zur Verfügung, welches Kontext, Dienste und kontextsensitive Mechanismen inkludiert. Aufgrund dieses Konzepts ist der Ansatz vor allem für die modellgetriebene Entwicklung von Anwendungen interessant, welche auf einem vom Systementwickler erstellten Modell beruhen. Für Sheng und Benatallah ergibt sich daraus der Vorteil, dass Produktivität und Qualität während der Entwicklungsphase enorm gesteigert werden können. Der größte Pluspunkt eines solchen modellgetriebenen Systems ist jedoch die Tatsache, dass die Implementierung von Diensten dabei nahezu automatisch generiert werden kann – sogar für verschiedene Plattformen.

Abbildung 2.8 zeigt das Metamodell, welches die abstrakte Syntax von ContextUML darstellt und von zwei Sichtweisen betrachtet werden kann – der Kontext-Modellierung sowie der Modellierung von Kontextsensitivität. Die erste Perspektive beinhaltet folgende Elemente des Metamodells:

- **Context:** Repräsentiert die Kontextdaten selbst und ist unterteilt in Low- (*AtomicContext*) und High-Level-Kontextinformationen (*CompositeContext*).
- **ContextSource:** Diese Klasse stellt die Quelle der Kontextdaten dar, wobei es wiederum zwei Typen gibt – *ContextService* und *ContextServiceCommunity*.

Neben diesen Elementen sind für die Modellierung von Kontextsensitivität weitere Module notwendig:

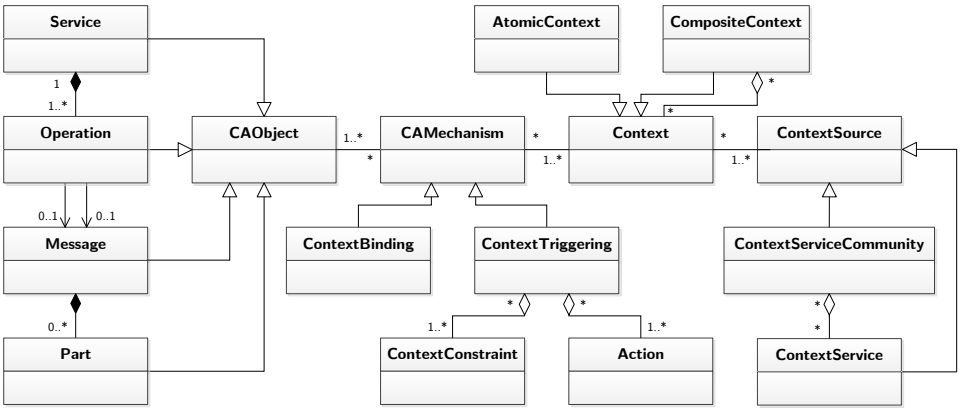


Abbildung 2.8: Metamodell von ContextUML (nach [53]).

- **CAMechanism:** Stellt ein Element dar, welches den Mechanismus der Kontextsensitivität formuliert. Ein Mechanismus kann Elementen des Typs *CAObject* zugeordnet werden und diese im weiteren Verlauf beeinflussen oder ändern.
- **CAObject:** Repräsentiert ein kontextsensitives Objekt – etwa eine Nachricht oder einen Dienst – welches situationsbedingt angepasst werden soll.
- **ContextBinding:** Als eine Art von *CAMechanism* erlaubt diese Klasse die automatische Zuordnung von Kontext zu kontextsensitiven Objekten.
- **ContextTriggering:** Eine weitere Art von *CAMechanism* ist das Modul *ContextTriggering*. Dies stellt die kontextsensitive Adaption von Objekten dar und inkludiert das Ausführen von Diensten aufgrund spezifischer Kontextinformationen. Dies wird mittels *ContextConstraints* und *Actions* erreicht, welche Bedingungen bzw. Aktionen einer bestimmten kontextsensitiven Adaption darstellen.

Auch wenn dieser Ansatz eigentlich für die Umsetzung kontextsensitiver Webdienste erarbeitet wurde, ist ein Einsatz auf anderen Gebieten durchaus denkbar, da es viele Bemühungen gibt, modellgetriebene Systeme zu entwickeln und dieses Projekt das erste seiner Art darstellt, welches auf die Umsetzung von kontextsensitiven Systemen mittels modellgetriebenen Architekturen (MDA – Model-Driven Architecture) eingeht.

2.3 Kontext-Adaption

Einen wichtigen Aspekt kontextsensitiver Systeme stellt die Adaption der Anwendung dar – also die Art und Weise wie sich die Applikation an die persönliche Situation des Benutzers anpasst. Dabei können etwa statisch definierte Bedingungen, dynamische Regel-Mechanismen oder Ontologien verwendet werden. Dieser Abschnitt präsentiert – chronologisch sortiert – eine Vielzahl an existierenden kontextsensitiven Systemen und beschreibt die Konzepte, welche verwendet werden, um die Anwendung – bezogen auf den teilweise oft wechselnden Kontext – zu adaptieren. Außerdem wird untersucht, welche Teile der Anwendung angepasst werden (z. B. Darstellung, Informationsgehalt, Verhalten etc.) und eine Auswertung hinsichtlich der verwendeten Konzepte vorgenommen.

2.3.1 Bestehende Anwendungen

ParcTab (1995)

Als eines der ersten kontextbezogenen Systeme gilt das von Want u. a. entwickelte ParcTab [61], das mit Hilfe von Tabs (Vorfahren der heutigen PDAs) und einem Infrarot-Netzwerk Nachrichten austauscht und Objekte lokalisiert. Mittels stationärer Device-Agents können an die Mobilgeräte verschiedene Anwendungen angebunden werden – etwa ein kontextsensitiver Datei-Browser. Dabei werden – abhängig von der aktuellen Position – nur jene Dateien des Dateisystems visualisiert, welche einen Bezug zum aktuellen Raum besitzen. Diese Relationen müssen bei Erstellung der Datei angegeben werden, um den Filtermechanismus der Anwendung korrekt arbeiten zu lassen.

C-MAP (1998)

Fels u. a. beschreiben in ihrem Artikel das Projekt C-MAP [21], einen kontextsensitiven Tour-Guide für Ausstellungen, Museen oder Messen. Die Anwendung nutzt zuvor angegebene Interessen des Benutzers sowie einen Filtermechanismus, um eine Tour durch eine Messe zu planen. Der Benutzer soll somit den Weg zu den für ihn interessanten Ausstellungspunkten finden.

Mit Hilfe einer physischen Karte wird der Benutzer auf dem Messegelände positioniert – eine semantische Kartendarstellung zeigt dem Anwender jene Aussteller, die sinngemäß nahe bei seinen eigenen Interessen liegen.

Conference Assistant (1999)

Um Teilnehmer an Konferenzen mit Hilfe von Kontextinformationen zu unterstützen, wurde von Dey u. a. der Conference Assistant [18] entwickelt. Diese Anwendung baut auf das Context Toolkit [17] auf und versucht etwa

– nach der Angabe von persönlichen Präferenzen mittels Benutzerprofil – für die jeweilige Person interessante Vorträge zu finden. Diese werden dem Benutzer mitsamt Detailinformationen präsentiert – es erfolgt also eine Adaption hinsichtlich Informationsgehalt.

Das Selektieren von passenden Vorträgen übernimmt ein sogenannter Kontext-Interpreter, welcher die Kontextinformationen (persönlichen Präferenzen sowie aktive Präsentationen) heranzieht und passende Themen herausfiltert.

GUIDE (1999)

Das für die Stadt Lancaster entwickelte System GUIDE [12] stellt einen mobilen, kontextsensitiven Stadtführer dar. Die webbasierte Anwendung verwendet als einzige Kontextinformation die aktuelle Position des Benutzers, um Informationen über Sehenswürdigkeiten in der näheren Umgebung anzuzeigen. Dabei wird die aktuelle Position in einen Anfrage-URL (Uniform Resource Locator) kodiert, welcher an einen Server geschickt wird. Die eigentliche Adaption der Informationen passiert innerhalb des Server-Dienstes, welcher die zur Position passenden Objekte identifiziert und an das Gerät retourniert.

News Dude (1999)

Den persönlichen Nachrichten-Agenten News Dude [7], welcher seinem Benutzer interessante Neuigkeiten bzw. Nachrichten präsentiert und vorliest, haben Billsus und Pazzani entwickelt. Anhand der Rückmeldung des Anwenders werden dessen Präferenzen und Interessen zu bestimmten Themenbereichen verwendet, um zukünftige Nachrichten nach eben diesen Interessen zu filtern. Die Rückmeldung erfolgt dabei verbal – etwa kann der User den News Dude beim Vorlesen eines Artikels unterbrechen und somit sein Desinteresse signalisieren. Des Weiteren kann der Benutzer nach Abhorchen einer Nachricht seine Einstellung zu diesem Thema andeuten – etwa wird ein vom User abgegebenes „interesting“ von der Spracherkennung des News Dude erkannt und weiterverwendet. Neue Artikel werden mit Hilfe dieses Wissens und der Artikel-Kategorie in einem eigenen Modul bewertet und nach Relevanz sortiert. Somit werden – nach einer Lernphase der Applikation – nur die für den Benutzer interessanten Themengebiete angesprochen.

CybreMinder (2000)

Dey und Abowd beschreiben in ihrem Artikel [16] ein kontextsensitives Tool, welches den Empfang bzw. die Erinnerung an Aufgaben oder Notizen situationsgerecht steuert. Dabei werden vom Benutzer bei Erstellung der Notiz Situationen mit Hilfe von Kontextattributen (z. B. Lokalität) definiert, in

denen die zugehörige Erinnerung erscheinen soll. Diese so erstellten, dynamischen Regeln können weitere kontextbezogene Daten berücksichtigen – etwa die Zeit, die aktuelle Temperatur oder Namen des Benutzers.

Das Projekt verwendet die Infrastruktur des Context Toolkits [17] und ermöglicht es, Erinnerungen an Aufgaben per SMS, E-Mail, Display-Anzeige (PDA, CRT etc.) oder in Papierform (Ausdruck) zu versenden.

ARREAL (2001)

Wahlster u. a. entwickelten im Rahmen des Projektes REAL [60] das ressourcenadaptierende Navigationssystem ARREAL – gedacht für Fußgänger im Outdoor-Bereich. Die Anwendung stellt eine Karte dar, welche die aktuelle Umgebung des Benutzers visualisiert. Das Kartenmaterial wird dabei nicht nur mit Hilfe der Position, sondern auch mittels der gegenwärtigen Benutzergeschwindigkeit und der gerade vorhandenen Ortsgenauigkeit adaptiert. So zeigt die Anwendung etwa bei niedriger Geschwindigkeit eine genauere, detailreichere Darstellung – bei hoher Geschwindigkeit wird ein größerer Kartenausschnitt mit höherem Zoomlevel gewählt.

Die Adaption der Karte ist hierbei durch eine einfache Formel direkt im Source-Code implementiert.

CRUMPET (2001)

Das von der EU initiierte Projekt CRUMPET [44] hat zum Ziel, benutzerfreundliche, mobile Dienste für den Tourismus zu entwickeln. Dabei werden vor allem der Positions- sowie der Benutzerkontext verwendet, um Informationen am Mobilgerät situationsgerecht anzupassen. Der Benutzerkontext wird automatisch und dynamisch generiert – es erfolgt keine Angabe der Interessen durch den Anwender. Mit Hilfe der so gewonnenen Präferenzen werden etwa POIs (Points of Interest) gefiltert, und nur die für den Benutzer interessanten auf einer Karte visualisiert.

HyperAudio & HIPS (2001)

Petrelli u. a. stellen in ihrem Artikel [42] die Erfahrungen aus der Implementierung von zwei positionsabhängigen adaptiven Museum-Guides (HyperAudio, HIPS) gegenüber. Beide Systeme ermöglichen es, die aktuelle Kontextsituation in die Applikationslogik einzubeziehen und somit auf die Einflüsse von außen zu reagieren.

Beim Besuch eines Museums wird – ausgehend von der Position des Anwenders – eine Präsentation der Objekte im näheren Umfeld erstellt und visualisiert. Diese besteht aus einer Audio-Mitteilung, einem Bild sowie weiterführenden Links. Die Präsentation wird passend zum aktuellen Kontext (Position des Benutzers) adaptiert, wobei dynamische Regel-Mechanismen eingesetzt werden. Nicht nur das Modul zur Erstellung von Präsentationen

sondern auch ein Input-Analyser verwendet Regeln, um das Verhalten der Anwendung zu steuern – etwa wird das Schließen der Präsentation bei Verlassen der jeweiligen Sehenswürdigkeit veranlasst.

PinPoint (2002)

Aufbauend auf der PinPoint-Infrastruktur wird von Roth ein kontextsensitiver Touristen-Führer beschrieben [45], welcher neben dem Gerätekontext auch die Position des Benutzers verwendet, um Informationen situationsgerecht darzustellen. Dabei werden die wichtigen Parameter (z. B. Koordinaten und Bildschirmauflösung) in einen URL kodiert – der serverseitige Dienst interpretiert diese Informationen und liefert das entsprechende Karten- und Informationsmaterial.

Chisel (2003)

Ein kontextsensitives, regelbasiertes Framework zur dynamischen Adaption von Applikationsverhalten [30] stellen Keeney und Cahill vor. Dieser Ansatz basiert auf der Verwendung von Systemen, welche zusätzlich zur üblichen Programmlogik auch Metadaten (Zusatzinformationen) über sich selbst besitzen. Darüber hinaus werden Änderungen in der Anwendung auch in den Metadaten präsentiert – umgekehrt ist es möglich, durch das Adaptieren von Metadaten das Verhalten der Anwendung zu steuern. Diesen Ansatz nutzt Chisel, um die Anwendung – ohne direkt in diese eingreifen zu müssen – in ihrem Verhalten kontextgerecht zu adaptieren.

Die Adaption erfolgt mit Hilfe dynamischer Regeln, welche außerhalb der Anwendung textbasiert definiert werden. Die Regeln enthalten üblicherweise einen Auslöser (z. B. ein bestimmter Event), ein Zielobjekt und eventuell Bedingungen, um die Regel auf bestimmte Situationen einzuschränken. Bei einem Wechsel der Kontextsituation werden alle verfügbaren Regelsätze abgearbeitet und bei Bedarf die zugehörige Aktion ausgeführt bzw. die Änderung der Metadaten veranlasst.

Gulliver's Genie (2003)

Einen weiteren mobilen kontextbezogenen Reiseführer für Touristen präsentieren O'Grady und O'Hare [40]. Dabei liegt der Fokus auf der Adaption von Präsentationen, welche bei Annäherung an eine Sehenswürdigkeit von einem zentralen Dienst geladen und visualisiert werden. Die Adaption des Präsentationsinhalts erfolgt auf Basis eines persönlichen Benutzerprofils sowie kultureller Interessen – beides wird vom Anwender im Vorhinein festgelegt. Mit Hilfe von Tags, welche den Elementen der Präsentation zugewiesen sind, kann der Adaptionsprozess für den User interessante Präsentationsinhalte selektieren bzw. filtern, und dadurch eine kontextsensitive Präsentation erstellen.

LOL@ (2003)

Umlauft u. a. beschreiben mit dem Projekt LOL@ [59] einen mobilen Touristenführer, erstellt für die Stadt Wien. Basierend auf dem Mobilfunkstandard UMTS (Universal Mobile Telecommunication System) wurde eine Anwendung entwickelt, welche es dem Benutzer u. a. erlaubt, Informationen über Sehenswürdigkeiten abzurufen, sich durch die Stadt navigieren zu lassen sowie ein elektronisches Tour-Tagebuch zu führen. Als einzige Kontextinformation wird dabei die Position des Benutzers verwendet, um auf einer Übersichtskarte nahe gelegene POIs visuell hervorzuheben. Diese Funktion wird durch einen zentralen Dienst realisiert, welcher die aktuellen GPS-Koordinaten per URL-Kodierung erhält, mit verfügbaren POIs abgleicht und der Anwendung einen Befehl zur besonderen Darstellung der nahe gelegenen Objekte sendet.

mobiDENK (2003)

Das Projekt mobiDENK [32] realisiert ein mobiles, ortsbezogenes Informationssystem und wurde von Krösche u. a. entwickelt. Die Anwendung basiert auf der Niccimon-Plattform [3] und dient dazu, Touristen auf nahe gelegene, besonders wertvolle Denkmäler und historische Sehenswürdigkeiten hinzuweisen sowie ortsabhängige multimediale Informationen dazu zu liefern. Mit Hilfe eines eingebauten GPS-Empfängers visualisiert ein Mobilgerät eine Karte mit der aktuellen Umgebung des Benutzers. Neben der eigenen Position werden von diesem Modul auch POIs – diese repräsentieren die Denkmäler und Sehenswürdigkeiten – dargestellt. Durch das Selektieren bzw. Filtern verschiedener Typen von Sehenswürdigkeiten kann der Anwender die Auswahl der angezeigten Objekte einschränken und auf seine eigenen Interessen begrenzen. Dadurch werden neben der Positionsinformation auch persönliche Interessen des Users in die berücksichtigten Kontextinformationen aufgenommen.

Sigtseeing4U (2003)

Boll beschreibt in ihrem Artikel Sightseeing4U [8], einen auf Webservices basierenden multimedialen Fremdenführer. Dabei werden primär Benutzer- und Gerätekontext verwendet, um multimediale Inhalte situationsgerecht auszuwählen und als Präsentation an den Client zu übertragen. Der Anwender wählt seine Präferenzen anhand einer Eingabemaske beim Start der Applikation. Wird eine Anfrage an den serverseitigen Dienst gestellt (z. B. „Sightseeingtour durch Wien“), so werden mittels der Interessen des Benutzers die interessantesten Objekte aus einer internen Datenbank gefiltert. Eine daraus verfasste personalisierte Präsentation wird dem User am mobilen Client visualisiert.

CASS (2004)

Das von Fahy und Clarke [20] entwickelte CASS-System (Context-Awareness Sub-Structure) ist eine Middleware für mobile kontextbezogene Anwendungen. Dabei wird ein Verfahren namens Forward-Chaining eingesetzt, um Adaptionen in der Anwendung abzubilden. Dieses nutzt dynamische Regeln, um von bekannten Fakten (z. B. Kontextinformationen) neue, auf höherer Ebene dargestellte Fakten (z. B. eine bestimmte Situation) abzuleiten. Darauf aufbauend können Anwendungsadaptionen definiert werden – etwa das Darstellen von aktuellen Kontextinformationen für den Benutzer oder das automatische Ausführen von Aktionen bzw. Diensten.

COMPASS (2004)

Der von van Setten u. a. beschriebene kontextsensitive persönliche Assistent (COntext-aware Mobile Personal ASSistant) [51] bietet Informationen und Dienste speziell für touristische Benutzer. Dabei werden sowohl der aktuelle Benutzerkontext als auch diverse Präferenzen des Anwenders berücksichtigt, um bei einer Anfrage (z. B. „Finde eine Übernachtungsmöglichkeit“) für den Anwender interessante Objekte (POIs) zu identifizieren. Die Adaption der Ergebnismenge übernimmt eine Recommendation-Engine, welche wiederum – mit Hilfe der Kontextdaten und der Benutzerpräferenzen – verschiedene Strategien zur Bewertung der POIs verwendet und die für den Anwender interessanten Informationen herausfiltert. Die gelieferten Ergebnisse werden dem Benutzer in Form einer Liste (mitsamt Beurteilung von anderen Usern) visualisiert.

m-ToGuide (2005)

Schwinger u. a. beschreiben in ihrem Artikel eine Menge mobiler Touristen-Guides, darunter auch den m-ToGuide [50]. Dieser verwendet Benutzerpräferenzen in Form von Profilen, um die für den Anwender interessanten POIs zu finden bzw. aus einer Gesamtmenge herauszufiltern. Nicht nur die POIs selbst, sondern auch deren Beschreibungen werden kontextsensitiv adaptiert. Um die Profile aktuell zu halten, wird das Verhalten des Benutzers untersucht und dessen Präferenzen automatisch aktualisiert. Neben dem Benutzerkontext spielt bei m-ToGuide auch der Zeitkontext eine Rolle – etwa werden Öffnungszeiten der POIs (z. B. Museen) berücksichtigt, um auf besuchbare Sehenswürdigkeiten hinzuweisen.

xPOI (2005)

Krösche und Boll verwenden in ihrem Projekt xPOI [33] Regelmechanismen (ECA-Regeln – Event-Condition-Action-Regeln), um kontextsensitive Informationen in die Anwendung zu integrieren. Dabei werden georeferenzierte

Interessenspunkte (POIs) in Form von Icons auf der Landkarte an die aktuelle Situation des Benutzers angepasst. Beispielsweise könnte die Darstellung von U-Bahn-POIs – also Teile der Benutzeroberfläche – aufgrund länderspezifischer Eigenheiten adaptiert werden.

Chameleon (2006)

Basierend auf dem URICA-Framework (Usage-awaRe Interactive Content Adaptation) wurde von Mohomed u. a. Chameleon [37] entwickelt – ein ressourcenadaptierendes System, das Bilder von Websites durch Adaption der Qualität an bandbreiten-limitierte Verbindungen anpasst. Dabei wird auf Client-Seite ein adaptiver Browser verwendet, welcher Rückmeldungen bzw. die Veränderung der Qualität von jedem Bild durch den Benutzer zulässt. Das vom Anwender somit erzeugte Feedback hinsichtlich Bildqualität wird von einem Proxy entgegengenommen, welcher die Webinhalte vom eigentlichen Server lädt, Adaptierungen durchführt und die angepassten Daten an den Benutzer weiterleitet. Alle Rückmeldungen werden des Weiteren von einem History-Mechanismus aufgezeichnet, welcher den zukünftigen Adaptierungsprozess beeinflusst.

ContextL/ContextJ (2006)

Costanza u. a. stellen in ihren Artikeln [14, 15] einen auf Layern basierenden Ansatz (ContextL) zur Anwendungsadaption vor. Abbildung 2.9 illustriert diese Idee – von einem bestimmten Kontext aus werden Methodenaufrufe getätigt. Der jeweils zuständige bzw. aufgrund der Kontextsituation aktive Layer übernimmt die Anfrage und führt die zugehörigen Aktionen aus.

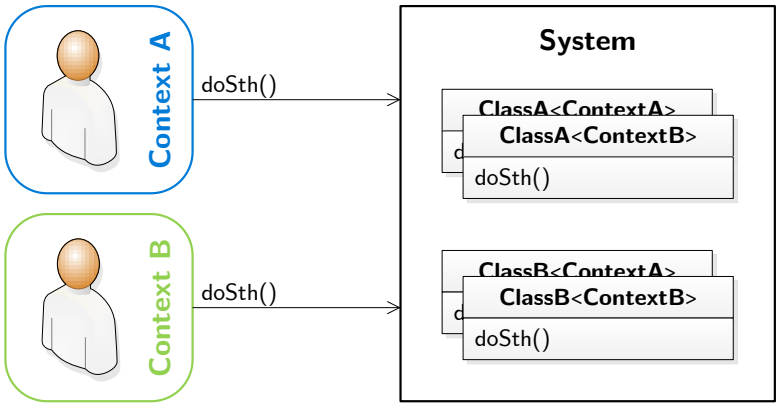


Abbildung 2.9: Layerbasierter Ansatz (nach [14]).

ContextL ist im Kern eine Erweiterung der Programmiersprache CLOS (Common Lisp Object System), welche wiederum auf der Verwendung von

generischen Funktionen basiert. Das für Java verfügbare Pendant ContextJ ist ebenso wie ContextL dazu gedacht, kontextorientierte Programmierung zu unterstützen und zu vereinfachen. Dies geschieht mit Hilfe von Layern bzw. Schichten, wobei ein Root-Layer existiert und beliebig viele weitere Layer definiert werden können. Jeder dieser Layer kann nun dynamisch von der Anwendung aktiviert bzw. deaktiviert werden. Um dem Layer Leben zu verleihen, können Klassen mit einem spezifischen Layer assoziiert werden. Dies bedeutet, es wird innerhalb des Layers eine Klasse, welche außerhalb bereits existiert, um Funktionen erweitert. Diese Funktionen werden nur dann ausgeführt, wenn der spezifische Layer aktiv ist. Somit ist es bspw. möglich, Zusatzinformationen, welche von einem dafür vorgesehenen Layer stammen, nur in bestimmten Situationen anzuzeigen.

MOPET (2007)

Der MObile PErsonal Trainier [10] ist eine von Buttussi und Chittaro entwickelte Anwendung, welche im Fitness-Bereich zum Einsatz kommt. Dabei werden – mit Hilfe von Sensoren – die sportlichen Aktivitäten einer Person analysiert und ausgewertet. Basierend auf dieser aktuellen Situation und persönlichen Daten (z. B. Alter) kann die Anwendung motivierende Anweisungen oder Alarme zu Sicherheit bzw. Gesundheit geben. Außerdem werden dem Anwender interaktive 3D-Animationen visualisiert, um die korrekte Ausführung der jeweiligen Übungseinheit zu zeigen.

Die kontextsensitive Adaption erfolgt hierbei durch das Vorschlagen von neuen Übungen bzw. durch alarmierende Hinweise, die Übung (z. B. laufen) zu verlangsamen – etwa wenn die Herzfrequenz zu hoch wird. Diese Entscheidungen werden mit Hilfe der aktuellen Sensorwerte sowie der Informationen über den Anwender (z. B. Geschlecht) getroffen und sind als Regeln in den Source-Code eingebettet.

Ubidoo (2007)

In ihrem Artikel beschreiben Stahl u. a. den positionsabhängigen Task-Planer Ubidoo [55]. Die Anwendung stellt eine kontextsensitive ToDo-Liste dar, welche – in Kombination mit einer Routenplanung – jene Tasks herausfiltert, die in der näheren Umgebung abgearbeitet werden können. Außerdem nimmt der webbasierte Ansatz Rücksicht auf den verfügbaren Zeitslot bis zum nächsten Fixtermin, die Öffnungszeiten von Geschäften und die benötigte Zeit, um den jeweiligen Ort zu erreichen. Um dies zu ermöglichen, werden die Tasks mit Orten (z. B. eine Adresse) oder Kategorien (z. B. Geschäftstyp) verknüpft, wobei das Ontologie-System UbisWorld [54] zum Einsatz kommt. Die mit Hilfe dieser Informationen durchgeführte, kontextabhängige Adaption bezieht sich auf die aktuelle Task-Liste und auszuführende Alarme. Dabei wird die aktuelle Position des Benutzers sowie die aktuelle Uhrzeit heran-

gezogen, um jene Tasks zu selektieren, welche im näheren Umkreis bzw. bis zum nächsten Fixtermin abgearbeitet werden können.

Die Selektion der Tasks erfolgt dabei in Kombination mit einem Routenplanungsalgorithmus. Dieser berechnet die geografischen Distanzen zwischen aktueller Position und der jeweiligen Task-Position sowie die benötigte Zeit, um dorthin zu gelangen. Ausgehend davon werden nur jene Tasks visualisiert, für die eine Bearbeitung in der aktuellen Situation sinnvoll erscheint.

Conny (2008)

Dürr u. a. beschreiben in ihrem Artikel [19] eine Möglichkeit, mit der das von Instant-Messaging-Diensten verwendete XMPP-Protokoll (Extensible Messaging and Presence Protocol) um kontextbezogene Adressierungskonzepte erweitert werden kann. Der entwickelte Prototyp basiert auf der Nexus-Plattform [38] und ermöglicht es, Nachrichten nur an Personen zu versenden, welche sich in einer definierten Kontextsituation befinden. Dabei versieht der Sender der Nachricht diese mit diversen Attributen bzw. Regeln, welche den gewünschten Kontext beschreiben. An einen Server übermittelt, wird die Nachricht von dort aus nur an jene Benutzer weitergeleitet, deren Kontext dem definierten entspricht.

Tacticycle (2009)

Das von Poppinga u. a. entwickelte Tacticycle [43] ist ein mobiles, kontextsensitives System, welches Touristen auf ihrer Fahrradtour per PDA begleitet und Orientierungshilfen bietet. Außerdem gibt die Anwendung dem Benutzer über zwei vibrationsfähige Griffe taktile Auskünfte – etwa um auf nahe gelegene Sehenswürdigkeit hinzuweisen.

Tacticycle verwendet keine dynamischen Regel-Mechanismen, sondern statisch definierte Bedingungen, um die kontextsensitiven Informationen an den Benutzer weiterzuleiten.

2.3.2 Auswertung

Die soeben vorgestellten praxisnahen Projekte bilden die Basis für die Auswertung hinsichtlich einiger wichtiger Aspekte, welche für den weiteren Verlauf dieser Diplomarbeit interessant und wichtig erscheinen:

- Welche Kontextparameter werden zur Adaption herangezogen?
- Welche Applikationsbereiche werden aufgrund der aktuellen Kontextsituation adaptiert?
- Wie erfolgt diese Anpassung?

Kontextparameter

Grundlage der Adaptionskonzepte sind diverse – vom System erfasste – Kontextparameter. Abbildung 2.10 zeigt eine Auflistung der von den vorgestellten Projekten verwendeten Kontextdaten (Details siehe Tabelle A.1 im Anhang).

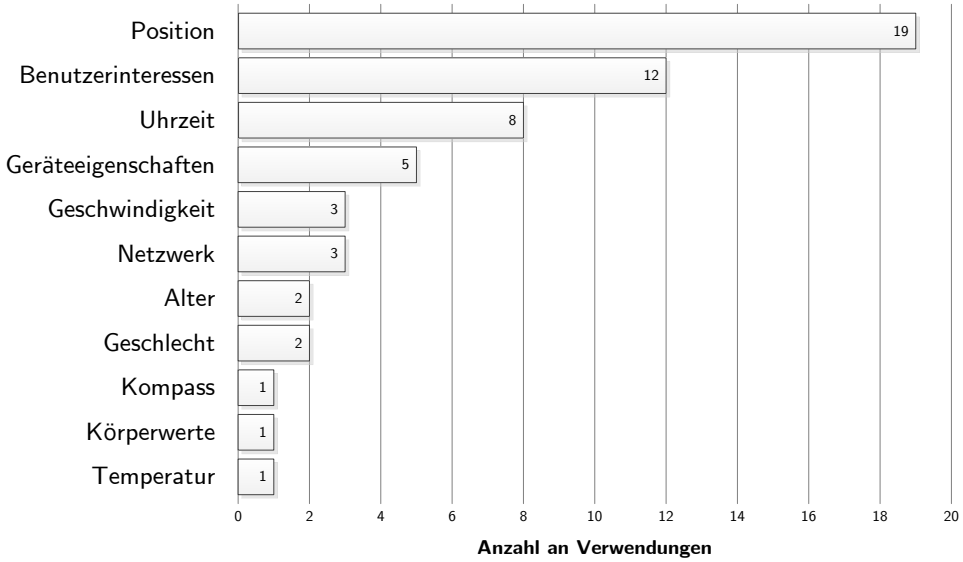


Abbildung 2.10: Verwendete Kontextparameter.

Es zeigt sich, dass der am häufigsten verwendete Kontextparameter die Positionsinformation ist. Dies könnte darauf zurückzuführen sein, dass bereits sehr viele Mobiltelefone einen integrierten GPS-Empfänger besitzen und es daher relativ komfortabel ermöglichen, positionsabhängige Anwendungen zu erstellen. Dicht dahinter folgen Kontextinformationen, welche – meist in Form eines Benutzerprofils abgefragt – die Interessen des Anwenders widerspiegeln. Damit können etwa Informationen, welche zu den bevorzugten Interessen des Benutzers zählen, anders visualisiert werden als der Rest der Daten. Ebenfalls notwendig ist diese Art der Kontextparameter für das Filtern von etwaigen Daten nach Interessensgebieten.

Weitere für kontextsensitive Applikationen wichtige Parameter sind Uhrzeit, Geräteeigenschaften, verfügbare Netzwerke sowie die aktuelle Geschwindigkeit. Damit ist es bspw. möglich, einen Tag- bzw. Nachtmodus zu unterstützen, Anpassungen hinsichtlich Displaygröße durchzuführen oder die Darstellung eines Kartenausschnitts zu optimieren.

Einzelne Anwendungen nutzen darüber hinaus Kompassdaten, Temperatur, Alter und Geschlecht des Benutzers sowie Körperwerte (Puls etc.). Meist werden solche selten verwendete Informationen für außergewöhnliche Applikationen benötigt – Körperdaten etwa für einen mobilen Fitnesstrainer.

Adaptionsarten

Die Anpassung der Anwendung, welche auf den erfassten Kontextparametern basiert, kann auf verschiedene Weisen erfolgen. Abbildung 2.11 zeigt eine Auswertung der Projekte hinsichtlich verwendeter Adaptionsart (Details siehe Tabelle A.2 im Anhang).

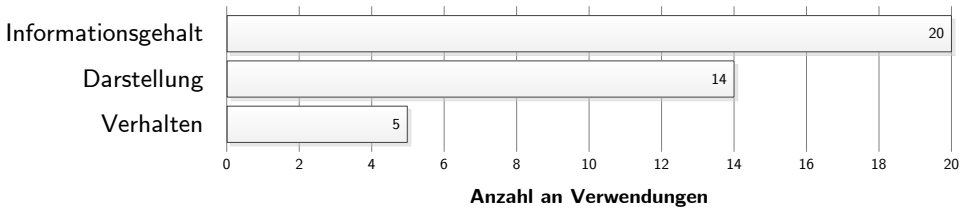


Abbildung 2.11: Verwendete Adaptionsarten.

Dabei ist ersichtlich, dass die häufigste Art, die aktuelle Situation in Anwendungen miteinzubeziehen, die Informationsadaption bzw. die Anpassung der dargestellten Daten ist. Dies könnte den Grund haben, dass dies technisch relativ einfach ist – z. B. können aus Textinformationen bestimmte Abschnitte entfernt werden. Die Darstellung wird ebenfalls von vielen Projekten hinsichtlich aktueller Situation angepasst – beispielsweise durch die Visualisierung des korrekten Kartenausschnitts. Grund für den relativ häufigen Einsatz dieser Adaptionsart könnten die für den Anwender sofort erkennbaren, positiven Veränderungen sein – bei der Informationsadaption etwa ist dies hingegen nicht der Fall.

Weniger oft wird das Verhalten der Applikation verändert. Denkbar ist, dass das Verhalten eher schwierig angepasst werden kann bzw. dies in den meisten der vorliegenden Projekte nicht unbedingt sinnvoll erscheint.

Neben der Vielzahl an praxisbezogenen Anwendungen werden zwei Projekte – Chisel und CASS – eher als Frameworks angesehen, um kontextabhängige Systeme zu entwickeln. Prinzipiell ist es mit beiden Systemen möglich, alle drei Arten der Kontext-Adaption umzusetzen.

Adaptionskonzepte

Die Anpassungen in Darstellung, Informationsgehalt und Verhalten können unterschiedlich erfolgen. Abbildung 2.12 zeigt jene Konzepte, welche die vorgestellten Projekte verwenden, um die Anwendung aufgrund der aktuell herrschenden Kontextsituation zu adaptieren (Details siehe Tabelle A.3 im Anhang).

Deutlich am häufigsten werden Filtermechanismen – meist in Kombination mit Benutzerprofilen – herangezogen, um die Adaption zu verwirklichen. Dabei kommen zum Teil auch lernende Methoden zum Einsatz, welche die Interessen der Anwender automatisch erfassen – als innovative Alternative

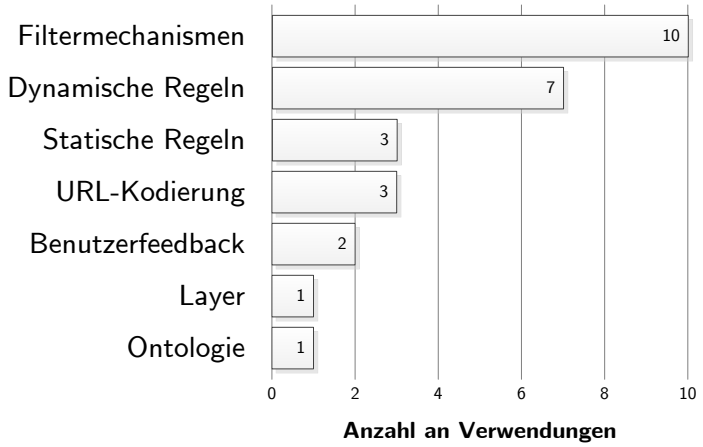


Abbildung 2.12: Verwendete Adaptionskonzepte.

zum ausfüllbaren Formular. Auffällig ist außerdem, dass jene Anwendungen, welche Filtermechanismen zur Adaption verwenden, hauptsächlich den Informationsgehalt der Applikation anpassen. Grund dafür könnte sein, dass Informationen relativ einfach nach bestimmten Kriterien (z. B. Interessenskategorien) gefiltert werden können.

Ebenfalls oft eingesetzt werden regelbasierte Mechanismen (statisch bzw. dynamisch), welche die Anwendung mit Hilfe von in den Source-Code eingebundenen bzw. in externen Dateien hinterlegten Bedingungen adaptieren. Ebenfalls mehrmals Verwendung findet URL-Kodierung – dabei wird jedoch neben der Client-Anwendung auch ein entfernter Dienst benötigt. All diese Konzepte sowie die weniger häufig eingesetzten Methoden (layer-, ontologie- und feedbackbasierte Ansätze) werden in den folgenden Abschnitten nochmals kurz erläutert.

2.3.3 Verwendete Adaptionskonzepte

Filtermechanismen

Eine relativ einfache Möglichkeit, um Anwendungen – besonders im Bezug auf Informationsgehalt – kontextsensitiv zu gestalten, stellen Filtermechanismen dar. Diese werden oft gemeinsam mit Benutzerprofilen verwendet, welche das Angeben von Interessen (z. B. per Formular) ermöglichen. Darüber hinaus gibt es im Gegensatz dazu auch lernende Systeme, welche die Präferenzen des Anwenders automatisch (über einen längeren Zeitraum) erfassen. Zur einfachen Weiterverarbeitung werden dem Benutzer oftmals definierte Werte (z. B. Interessensauswahl anhand bestehender Liste) vorgegeben.

Die Adaption selbst erfolgt nach dem Erfassen der Informationen mit Hilfe von Filtern. Ein Beispiel hierzu wäre die Adaption von Nachrichten

auf Basis des Benutzerinteresses. Dazu würden die Informationen mit den passenden, über das Profil auswählbaren Bereichen verknüpft. Bei der Abfrage neuer Nachrichten können sodann nur jene Nachrichten selektiert und visualisiert werden, welche zum Benutzerprofil passen.

Dynamische Regeln

Dieses Adaptionskonzept bezieht sich auf oft außerhalb der Anwendung (z. B. in externen Dateien) definierte kontextsensitive Bedingungen. Ein solches Konzept beschreiben Beer u. a. [5] – dabei werden sogenannte ECA-Regeln (Event-Condition-Action-Regeln) verwendet, um auf wechselnde Kontextsituationen einzugehen. Bei der Änderung von Objektattributen, welche Kontextinformationen enthalten, werden Events ausgelöst, welche mit Regeln in Verbindung stehen. Somit kann auf Kontextänderungen relativ einfach reagiert werden. Die Idee, auch zur Laufzeit neue Regeln in die Anwendung aufzunehmen, verleiht dem Ansatz zusätzliche Dynamik. Ein Beispiel für eine Regeldefinition beinhaltet Listing 2.1.

```
1 rules for <HeartPatients> {
2   on HeartMonitor.Alarm() {
3     <EmergencyDispatcher>.EmergencyCall:
4       Alarm(Patient);
5   }
6 }
```

Listing 2.1: ECA-Regel-Beispiel.

Dabei würde für jedes Objekt vom Typ `HeartPatient` bei einem auftretenden `HeartMonitor-Alarm` ein Notruf abgesetzt werden.

Statische Regeln

Anders als dynamische sind statische Regeln nicht außerhalb der Anwendung definiert, sondern direkt in den Source-Code eingebettet. Diese Bedingungen können als simple `if`- oder `switch`-Ansätze implementiert sein und sind – aufgrund der statischen Eigenschaft – relativ schwierig anzupassen bzw. zu erweitern. Dabei muss die gesamte Anwendung neu kompiliert und ausgeliefert werden.

URL-Kodierung

Um Daten bereits vorselektiert von einem Dienst abzurufen, kann der Ansatz der URL-Kodierung verwendet werden. Dabei wird der Aufruf an den Server-Dienst mit Parametern (z. B. die aktuelle GPS-Position) versehen. Der Dienst selektiert aufgrund dieser zusätzlichen Angaben die gewünschten

Daten (z. B. den umliegenden Kartenausschnitt) und retourniert das Ergebnis an den Aufrufer. Somit wird die Adaption der Informationen nicht direkt am Client durchgeführt, sondern an einen externen Dienst verlagert.

Benutzerfeedback

Weniger häufig Verwendung finden Mechanismen, welche das Feedback des Anwenders nutzen, um die Adaptionen durchzuführen. Einzig zwei Projekte (Chameleon, News Dude) nutzen dieses Konzept, wobei Rückmeldungen des Benutzers von der jeweiligen Applikation entgegengenommen und herangezogen werden, um direkt den Adaptionsprozess zu beeinflussen.

Layer

Architekturorientierte Ansätze, welche Layer zur Adaption von Anwendungen verwenden, werden ebenfalls selten eingesetzt. Im Rahmen dieser Arbeit konnte nur ein Projekt (ContextL) gefunden werden, welches nach diesem Prinzip arbeitet. Dabei werden die zu behandelnden Kontextsituationen in verschiedene, klar abgetrennte Teilbereiche der Applikation (Layer) aufgeteilt. Nur der aktive Layer erhält die vom Benutzer gestellte Anfrage und verarbeitet diese bzw. gibt eine Ergebnismenge zurück.

Ontologie

Ebenso nur eines der vorgestellten Projekte (Ubidoo) nutzt Ontologien zur Kontext-Adaption. Diese werden verwendet, um Relationen zwischen Aufgaben und Orten bzw. Aufgaben-Kategorien herzustellen. In weiterer Folge ist es möglich, in bestimmten Situationen nur jene Aufgaben auszuwählen bzw. zu visualisieren, welche zu diesem Zeitpunkt relevant sind.

2.3.4 Ergebnisse

Die Auswertung der existierenden Projekte zeigt, dass es diverse Ansätze gibt, um Anwendungen an die aktuelle Situation anzupassen. Ebenfalls erkennbar ist, dass keiner der Ansätze Statecharts zur Adaption nutzt, wenngleich dieses Konzept einige Vorteile mit sich bringt – etwa die Möglichkeit der visuellen Darstellung komplexer Anwendungslogiken. Um zu bestätigen, dass Anwendungsadaptionen auch mittels Statecharts möglich sind, stellt die vorliegende Arbeit in Kapitel 4 einen dazu passenden Ansatz sowie dessen Vorteile gegenüber bereits bestehenden Adaptionskonzepten vor.

Kapitel 3

Basistechnologien

Um kontextbezogene Aspekte mit Hilfe von Statecharts umsetzen zu können, werden diverse Basistechnologien benötigt. Dazu zählen Statecharts, welche bereits im ursprünglichen Lomotain-Projekt verwendet werden, um die interne Logik von Spielszenarien abzubilden. Darüber hinaus wird dieses Konzept eingesetzt, um das Ziel dieser Arbeit – einen bislang nicht existierenden, auf Statecharts beruhenden, Ansatz zur Anwendungsadaption – zu erreichen.

Neben der Konzeption dieses neuen Ansatzes ist außerdem die Entwicklung einer Authoring-Anwendung notwendig, welche das Erstellen von Spielszenarien inklusive Statechart-Logik erlaubt. Als Basis dieser Applikation wurde die Eclipse RCP (Rich Client Platform) gewählt, welche den besonderen Vorteil besitzt, einzelne Anwendungsbestandteile (z. B. Statechart-Editor) vollautomatisch generieren zu können.

Für die mobile Client-Anwendung, welche das Ausführen der Spielszenarien sowie die Interaktion mit dem Benutzer übernehmen soll, wird u. a. eine Technologie zum Übertragen der Spielbeschreibung benötigt. Aufgrund der zu Statecharts relativ ähnlichen Hierarchiestruktur wird hierbei XML (Extensible Markup Language) verwendet.

Ebenfalls im Rahmen der mobilen Applikation essenziell ist ein Konzept zum Parsen kontextsensitiver Bedingungen, welche – vom Entwickler in der Statechart-Logik definiert – die situationsabhängige Adaption der Anwendung steuern sollen. Zur Verarbeitung dieser Regeln wird, u. a. aufgrund der optimalen Kompatibilität mit der verwendeten Programmiersprache, JavaCC (Java Compiler Compiler) eingesetzt.

All die soeben genannten Technologien werden in den folgenden Abschnitten genauer vorgestellt. Des Weiteren wird dabei Bezug zum vorliegenden Projekt hergestellt um zu zeigen, warum sich diese Technologien hierfür besonders eignen.

3.1 Statecharts

Statecharts (Zustandsdiagramme) [6,46] sind ein in der Softwareentwicklung oft eingesetztes Werkzeug, um mit Hilfe von States (Zuständen), Transitions (Zustandsübergängen) und Actions (Aktionen) das Verhalten von Systemen oder Anwendungen darzustellen. Wesentlichen Anteil an der Entwicklung von Statecharts hat Harel [26] mit seinem Ansatz, welcher als Grundlage der aus der UML (Unified Modeling Language) bekannten Statecharts [39] gilt und bereits wichtige Aspekte (Hierarchie, Orthogonalität) beinhaltet.

Als Basis nutzen Statecharts Aspekte endlicher Automaten, wobei es zwei wichtige Ausprägungen davon gibt: Bei **Mealy-Automaten** hängt die Ausgabefunktion einerseits vom aktuellen inneren Zustand des Automaten ab, andererseits jedoch auch von den aktuellen Eingabewerten. Im Gegensatz dazu wird bei **Moore-Automaten** nur der aktuelle interne Zustand für die Ausgabe berücksichtigt. Außerdem werden hierbei Aktionen nur mit States assoziiert – nicht jedoch mit Transitions. Beide Ausprägungen werden im Rahmen der Statecharts verwendet – ersterer bei Actions, welche vom aktuellen Zustand und einem eingehenden Event (Eingabewert) abhängen. Entry- bzw. Exit-Actions von Statecharts verwenden wiederum die Eigenschaft des Moore-Automaten, Actions nur mit States – nicht aber mit Transitions – zu verbinden.

Gegenüber anderen Darstellungsarten bieten Statecharts eine Reihe an Vorteilen. Dazu gehören die Verringerung (durch Wiederverwendung bzw. Vererbung von Verhalten) und die Abstraktion von Komplexität – States können weitere (auf den ersten Blick nicht sichtbare) Substates enthalten. Dadurch kann ein modelliertes System auf diversen Abstraktionsebenen dargestellt werden. Ebenfalls erlauben Statecharts das Zerteilen von komplexen Systemen in modulare Teilbereiche, wodurch Lesbarkeit und Verständlichkeit stark erhöht werden. Weitere interessante Eigenschaften sind das Verpacken von States in einen übergeordneten Zustand und die Verwendung von Orthogonalität bzw. Nebenläufigkeit.

Im Rahmen der vorliegenden Arbeit werden Statecharts dazu verwendet, die interne Anwendungslogik von Spielszenarien darzustellen. Entwickler haben somit die Möglichkeit, durch simples Modellieren von States, Transitions und Actions ein neues Anwendungsszenario zu erstellen, welches von der Applikation interpretiert und ausgeführt wird. Neben diesem Anwendungsgebiet werden Statecharts in vielen weiteren Bereichen der Softwareentwicklung verwendet – etwa zur Visualisierung von Lebenszyklen (Verhalten der Anwendung über die Zeit). Darüber hinaus gibt es diverse Tools, welche die automatische Code-Generierung auf Basis von Statecharts erlauben (z. B. Erzeugung von Java-Klassen).

Abbildung 3.1 zeigt ein Beispiel eines Statecharts – die folgenden Abschnitte beschreiben nun die wichtigsten Elemente und Konzepte dieser Visualisierungsform.

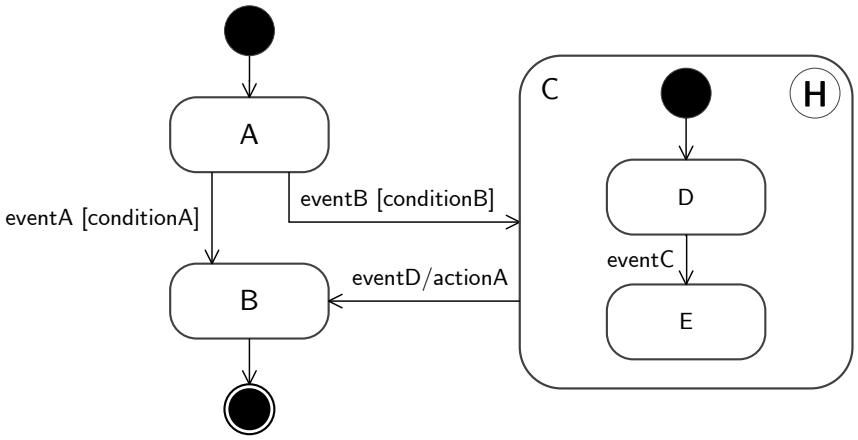


Abbildung 3.1: Statechart-Beispiel.

States (Zustände) – dargestellt durch abgerundete Rechtecke – stellen eine bestimmte Situation des Systems dar, in der es sich zu einem diskreten Zeitpunkt befinden kann. Einfache States (State A in Abbildung 3.1) haben – im Gegensatz zu zusammengesetzten States (State C) keine weiteren Subelemente, sondern nur ausgehende und eingehende Transitions sowie Actions und Activities. Zusammengesetzte States besitzen als Kindelemente weitere Zustände und somit eine hierarchische Struktur. Vorteil dabei ist, dass Eigenschaften – z. B. Transitions – des übergeordneten States (C) auch ausgeführt werden, wenn sich das System in einem Subzustand (z. B. D) befindet. Diese per Vererbungsmechanismus realisierte Funktionalität reduziert die Anzahl der notwendigen Transitions deutlich.

Transitions (Zustandsübergänge) werden durch gerichtete Graphen zwischen States visualisiert und besitzen neben Events (Trigger) auch Guards und Actions (Syntax: *Event* [*Guard*] / *Action*). Transitions stellen die Beziehungen zwischen zwei Zuständen her und spezifizieren den Zeitpunkt des Zustandswechsels, welcher durch einen eingehenden Event ausgelöst wird. Dabei werden Exit-Action des Quell-States, Transition-Action sowie Entry-Action des Ziel-States in dieser Reihenfolge ausgeführt. Wird zwischen zwei States unterschiedlicher Hierarchiestufen gewechselt, wird der kleinste gemeinsame Vorfahre bzw. Superstate (LCA – least common ancestor) der beiden Zustände berechnet. Für dessen Substates werden – ausgehend vom aktuell aktiven Zustand – die Exit-Actions ausgeführt. Gegensätzlich dazu (von höchster zu niedrigster Ebene) werden danach die Entry-Actions aufgerufen.

Zwei besondere Konzepte stellen interne sowie Self-Transitions dar. Erstere führen zu keinem Zustandswechsel, sondern zu einem Aktionsaufruf bei Eintreten des jeweiligen Events. Bei Self-Transitions (haben den selben Quell- und Ziel-State) werden hingegen Exit- und Entry-Actions ausgeführt.

Guards (Bedingungen) werden im Rahmen von Transitions spezifiziert und müssen zum Zeitpunkt des Zustandswechsels *true* ergeben, um diesen durchzuführen. Ergibt die Bedingung *false*, so wird der Zustandswechsel abgebrochen.

Actions (Aktionen) sind an Transitions oder States gekoppelte Operationen (z. B. Funktionsaufrufe). Dabei gibt es Entry- sowie Exit-Actions für Zustände, welche vor bzw. nach einem Zustandswechsel ausgeführt werden. Zustandsübergänge hingegen besitzen nur eine einzige Action, welche bei erfolgreichem Wechsel der States ausgeführt wird.

Activities (Aktivitäten) sind länger andauernde Actions. Damit können etwa innerhalb eines bestimmten Zeitraums (z. B. solange sich das System in State A befindet) gewünschte Tätigkeiten durchgeführt werden.

Pseudostates stellen besondere, unvollständige Zustände dar, da diese keine Actions bzw. Activities besitzen. Es gibt dabei drei wichtige Arten: Initial-States (schwarzer Punkt) definieren den Startzustand eines Statecharts, Final-States (schwarzer Punkt mit umschließendem Kreis) den Endzustand. History-States (H mit umschließendem Kreis) bewirken, dass das System bei Wiedereintritt in einen zusammengesetzten Zustand (z. B. State D in Abbildung 3.1) automatisch in den zuletzt aktiven Substate gelangt.

Orthogonale Regionen sind durch strichlierte Linien getrennte Bereiche von zusammengesetzten States. Bei Eintritt in den Superstate werden alle orthogonalen Regionen gleichzeitig betreten – dies bedeutet, es sind mehrere States (je einer aller orthogonalen Bereiche) gleichzeitig aktiv. Besonders hilfreich ist dieses Konzept, wenn das System in mehrere, gleichzeitig auszuführende Subsysteme geteilt werden soll.

Behavorial Inheritance beschreibt die bereits angesprochene Vererbung von Verhalten. Substates einer Zustandshierarchie erben damit bestimmte Eigenschaften der Superstates. Große Vorteile bringt dieses Konzept etwa bei Transitions mit sich: Die im Beispiel (Abbildung 3.1) abgebildeten States D und E würden ansonsten jeweils eine Transition zum State B benötigen. Durch das Konzept der Behavorial Inheritance wird somit Komplexität vermindert und Lesbarkeit deutlich erhöht.

3.2 Eclipse

Die soeben beschriebenen Statecharts sollen der Definition von Anwendungslogik für Spielszenarien dienen. Die Umsetzung einer Anwendung zum Erstellen dieser Anwendungslogik bzw. gesamter Spielszenarien (Logik und Inhaltsinformationen) kann auf unterschiedliche Arten erfolgen. Die im Rahmen dieser Arbeit verwendete Möglichkeit nutzt ein vorhandenes Framework, welches diverse Vorteile mit sich bringt und von der Open-Source-Community Eclipse Foundation[1] entwickelt sowie gewartet wird. Diese gemeinnützige

[1] http://www.eclipse.org

Gesellschaft stellt neben der primär für Java eingesetzten, frei verfügbaren Entwicklungsumgebung Eclipse IDE (Integrated Development Environment) zahlreiche Erweiterungen dazu bereit (z. B. Tools zur Anwendungsentwicklung für mobile Endgeräte). Ebenfalls bietet die Eclipse Foundation Softwareentwicklern viele Frameworks und Tools, mit deren Hilfe die Erstellung von Anwendungen erleichtert werden soll.

Zwei im Folgenden näher beschriebene Frameworks sind die Rich Client Platform und das Modeling Framework. Diese werden im Rahmen dieser Arbeit zur Erstellung einer Authoring-Anwendung verwendet, welche die Definition von Anwendungslogik (auf Basis von Statecharts) sowie das komplette Erstellen von Spielszenarien für das Lomotain-Projekt ermöglicht.

3.2.1 Rich Client Platform

Rich Clients sind Anwendungen, welche lokal auf dem Rechner des Anwenders arbeiten, bei Bedarf über ein Netzwerk mit anderen Applikationen (z. B. auf einem Server) kommunizieren und meist für eine bestimmte Anwendungsdomäne bestimmt sind [34]. Im Gegensatz zu Thin Clients (webbasiert) werden dabei alle Berechnungen am lokalen Computer durchgeführt und Komponenten des Host-Betriebssystems – etwa Objekte für die grafische Benutzeroberfläche (UI – User Interface) – genutzt. Im Laufe der Jahre entstanden Frameworks für die Erstellung von Rich Clients, welche die Programmierung dieser Anwendungen stark vereinfacht haben. Etwa übernehmen solche Frameworks eine Vielzahl an Aufgaben, die jede Anwendung benötigt – z. B. Teile des UI-Designs oder Datenbankanbindungen. Damit können Softwareentwickler den Fokus auf die Anwendungsdomäne legen und trotzdem einfach zu benutzende, funktionale Anwendungen erstellen.

Ein solches Framework zur Erstellung von Rich Client-Anwendungen ist die Eclipse RCP (Rich Client Platform) [34]. Die wohl bekannteste der zahlreichen Applikationen, welche auf dieser generischen Plattform basieren, ist die Eclipse IDE. Jedoch gibt es eine Menge weiterer Anwendungen – darunter ein von der US-amerikanischen Raumfahrtbehörde NASA entwickeltes Roboter-Steuerungstool – deren Basis die Eclipse RCP darstellt.

Für Entwickler bietet die Eclipse RCP einige nützliche Features – darunter ein natives UI-Toolkit, Hilfe- und Updatemechanismen, Möglichkeiten zur Erweiterung der Anwendung sowie Fehlerbehandlungsmechanismen. Außerdem ist das System sehr portabel – damit erstellte Anwendungen können auf unterschiedlichen Betriebssystemen und Hardware-Technologien verwendet werden. Einzig eine installierte JVM (Java Virtual Machine) mitsamt Java ME-Foundation-Bibliotheken (oder höher) wird vorausgesetzt. Abbildung 3.2 gibt einen Überblick über die Eclipse RCP sowie deren Verwendbarkeit für eigene Anwendungen.

Aufbauend auf die Java Virtual Machine bildet die Eclipse RCP die Basis für diverse Anwendungen – darunter eine generische IDE-Plattform sowie ei-

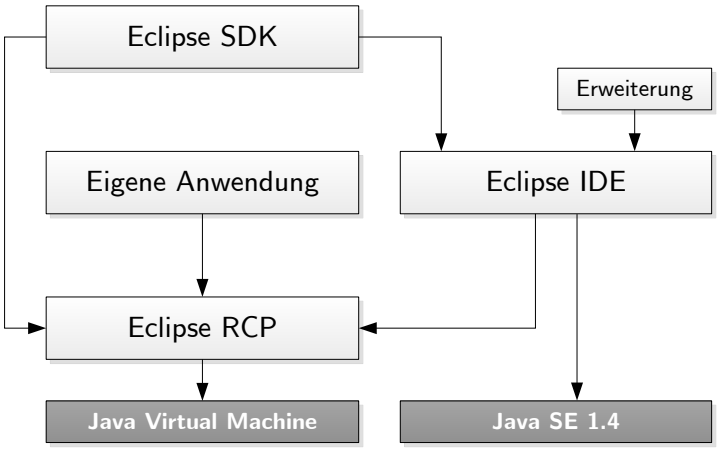


Abbildung 3.2: Eclipse RCP – Überblick (nach [34]).

gene RCP-Anwendungen. Diese können durch Erweiterungen um Funktionen ergänzt werden. Eine besonders bei Java-Entwicklern bekannte Komponente ist das Eclipse SDK, welches auf RCP und der generischen IDE-Plattform aufbaut und die bekannte Eclipse-Programmierumgebung darstellt.

Einzelne Funktionseinheiten von Eclipse RCP-Anwendungen definieren sogenannte *Plug-Ins*, welche aus mehreren Dateien (diese enthalten u. a. Source-Code und allgemeine Informationen) bestehen. Für eine Applikation benötigte Plug-Ins können selbst entwickelt oder von der Community bzw. von Firmen bezogen werden. Neben einer Sammlung dieser Anwendungsbauteile benötigen RCP-Programme Komponenten aus der Eclipse RCP, welche in Abbildung 3.3 dargestellt sind.

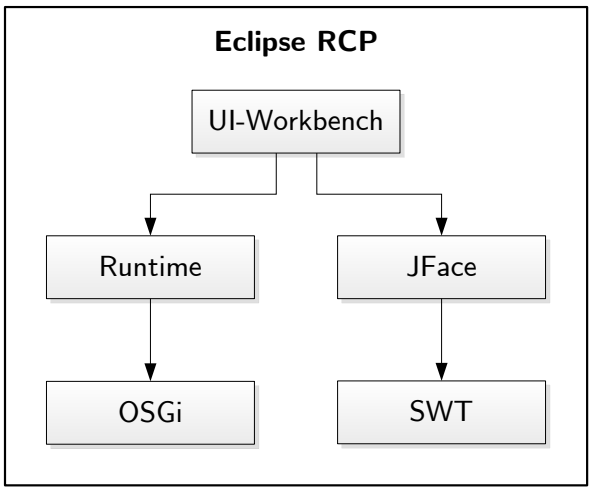


Abbildung 3.3: Komponenten der Eclipse RCP (nach [34]).

Eines der fünf enthaltenen Pakete stellt das **OSGi-Framework** (Open Services Gateway initiative) dar, welches in der Eclipse RCP für die modulbasierte Aufteilung der Anwendung in Plug-Ins verantwortlich ist. Die OSGi-Komponente führt innerhalb einer RCP-Anwendung installierte Plug-Ins zusammen und ermöglicht die Kommunikation zwischen diesen Modulen.

Aufbauend auf den OSGi-Layer wartet die **Runtime**, ähnlich wie die JVM, auf Anweisungen (in diesem Fall auf RCP-Anwendungen bzw. sogenannte Applications), um diese auszuführen. Daneben beinhaltet die Runtime eine Extension-Registry, in welcher die Erweiterungen spezifiziert werden. Interessant dabei ist, dass die Erweiterungen bei der Basis-Applikation nicht bekannt sein müssen – es ist somit möglich, bestehende Anwendungen ohne deren explizites Wissen um Funktionen zu erweitern.

Das **SWT** (Standard Widget Toolkit) ist eine portable, für viele Betriebssysteme verfügbare Grafikbibliothek, welche oft benötigte UI-Komponenten (sogenannte Widgets wie z. B. Listen, Buttons etc.) zur Verfügung stellt. Hauptvorteil von SWT ist die Verwendung der nativen, vom jeweiligen Betriebssystem zur Verfügung gestellten UI-Elemente. Damit entsteht eine für den Anwender bekannte grafische Oberfläche, welche darüber hinaus auch so reagiert wie dieser es gewohnt ist. Basierend auf SWT stellt **JFace** oft benötigte Konzepte und Strukturen grafischer Benutzeroberflächen bereit. Dazu zählen etwa Dialogboxen, Wizards, Views, Windows und Actions.

Diese vier soeben vorgestellten Komponenten bilden die Basis der **UI-Workbench** und somit die essenziellen Komponenten jeder Eclipse RCP-Anwendung. Entwicklern bietet die UI-Workbench zwei wichtige Funktionen – einerseits das deklarative Erweitern von Anwendungen um neue UI-Elemente (z. B. Menüeinträge) mit Hilfe von Extension-Points. Andererseits werden drei wichtige UI-Konzepte zur Verfügung gestellt – Editors (für Daten), Views (für Zusatzinformationen) und Perspectives (zur Anordnung von Views und Editors).

3.2.2 Modeling Project

Die modellbasierte Beschreibung eines Systems – etwa mit Hilfe von UML-Klassendiagrammen – ist heutzutage fixer Bestandteil eines Softwareentwicklungsprozesses. Neben der visuellen Präsentation eines Systems auf diese Weise ermöglichen diverse Modellierungs-Frameworks das Generieren von Source-Code (MDSD – Model Driven Software Development). Dieser Aspekt soll im Rahmen einer Authoring-Anwendung zur Erstellung eines Statechart-Editors genutzt werden. Grundlage der Technik ist die Definition eines Modells, welches im weiteren Verlauf in andere Modelle bzw. Darstellungsformen transformiert werden kann bzw. zur Erzeugung von Source-Code dient.

Vorteile, welche sich aus der modellgetriebenen Softwareentwicklung ergeben, sind etwa die Optimierung der Produktivität – auf Knopfdruck lassen sich vollständige Anwendungen generieren und somit viel Arbeit und

Zeit sparen. Der so generierte Code ist darüber hinaus effizient, einfach und frei von Fehlern, welche bei herkömmlicher Entwicklung durch Programmierer entstehen können. Dieses automatische Erstellen von Anwendungen darf jedoch nur als Hilfestellung für den Entwickler angesehen werden – i. d. R. müssen auf diese Weise erstellte Applikationen vom Programmierer erweitert bzw. angepasst werden, um das gewünschte Ergebnis zu erhalten.

Ein Softwarepaket, welches unter anderem diese Funktionen anbietet, ist das Eclipse Modeling Project [25, 56] – eine Sammlung von Tools und Frameworks rund um das Thema Modeling bzw. MDSD. Vorteil dieses Pakets ist die nahtlose Integration in die Eclipse IDE, welche alle wichtigen Komponenten des Modeling Projects als Plug-Ins aufnehmen kann. Den Kern des Modeling Projects stellt das **EMF (Eclipse Modeling Framework)** dar, welches eine Vielzahl an Modeling-Funktionen bereitstellt – z. B. Model-Transformation und -Validierung sowie Code-Generierung. Damit ist es etwa möglich, anhand eines zuvor definierten Modells (Domain-Model) einen auf RCP basierenden Texteditor zu erstellen, welcher das Erfassen genau der im Modell spezifizierten Objekte bzw. Strukturen und somit die Erstellung einer sogenannten Modell-Instanz erlaubt. Beim automatischen Erstellen eines solchen Editors werden mehrere Eclipse-Projekte angelegt (`model`, `edit` und `editor`), welche Eclipse Plug-Ins sind und gemeinsam die Anwendung verkörpern.

Neben EMF ist auch das darauf aufbauende **GMF (Graphical Modeling Framework)** Bestandteil des Modeling Projects. Diese Komponente ermöglicht das grafische Darstellen von Modellen, welche dadurch für Menschen einfacher zu verstehen sind. GMF besteht aus zwei Hauptkomponenten – einer Runtime, welche die Basisframeworks EMF und GEF (Graphical Editing Framework) verbindet sowie einer Tooling-Komponente. Diese bietet die Möglichkeit, grafische Elemente und eine Tooling-Palette zu definieren sowie die Zuordnung zwischen diesen Elementen und den Objekten im Domain-Model herzustellen. Ähnlich wie bei EMF ist auch mit GMF die Erstellung von Editoren – diesmal jedoch mit grafischer Visualisierung – realisierbar. Dabei wird ein neues Eclipse-Projekt (`diagram`) erstellt, welches als Plug-In den gesamten grafischen Editor beinhaltet und die zuvor mit EMF generierten Projekte referenziert.

Im Rahmen dieser Arbeit wird das Modeling Project dazu verwendet, einen grafischen Editor zur Definition von Statecharts zu erstellen. Der hierfür übliche Arbeitsablauf ist in Abbildung 3.4 dargestellt.

In einem ersten Schritt ist es notwendig, ein sogenanntes *Ecore-* bzw. *Domain-Model* (ähnlich zu UML-Klassendiagrammen, jedoch nicht so umfangreich) zu erstellen. Dieses kann per Editor angelegt oder aus vorhandenen Quellen (UML-Diagramme, XML-Schema-Spezifikationen oder Java-Interfaces) importiert werden. Auf Basis dieses Modells wird nun ein *Generator-Model* erstellt, welches für den ersten von zwei Code-Generierungs-Schritten notwendig ist. Das Generator-Model referenziert dabei das Do-

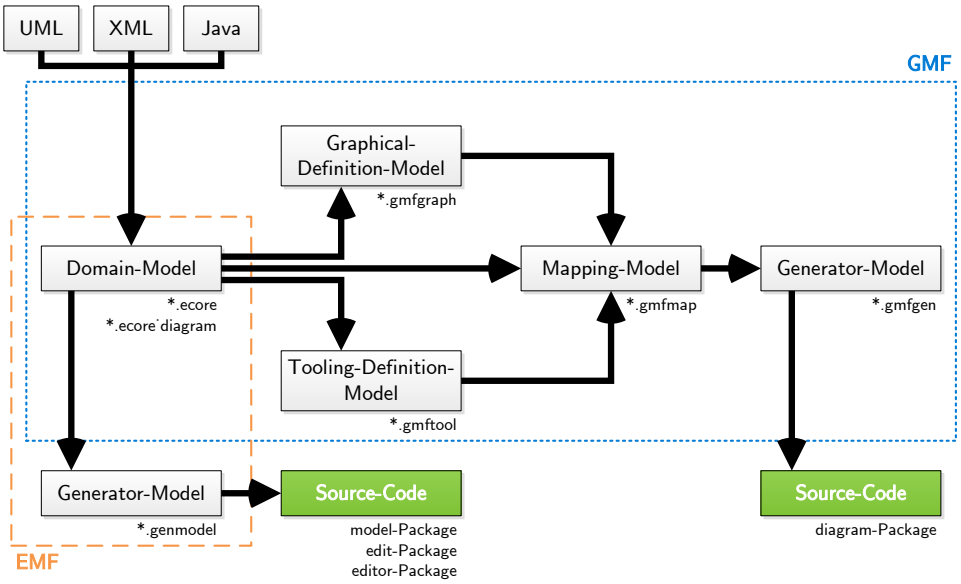


Abbildung 3.4: Workflow zur Erstellung eines grafischen Editors mit GMF.

main-Model und erweitert dieses um Informationen zur Code-Generierung, welche sodann angestoßen werden kann. EMF erzeugt dabei mehrere neue Projekte, darunter ein `editor`-Projekt. Dieses stellt den textbasierten Editor für das anfangs erstellte Domain-Model dar und kann bereits als eigene Instanz gestartet werden.

Um den eigentlich gewünschten grafischen Editor generieren zu lassen, sind einige Zwischenschritte erforderlich – etwa das Erstellen eines *Graphical-Definition-Models*. Dieses beinhaltet die grafischen Elemente, welche später im Editor zu sehen sind und die Objekte des Domain-Models darstellen. Des Weiteren wird ein *Tooling-Definition-Model* benötigt, welches eine Palette für das Erstellen von neuen Objekten festlegt.

Ein *Mapping-Model* referenziert die drei zuletzt erzeugten Dateien und stellt Verknüpfungen zwischen den einzelnen grafischen Elementen und den Objekten des Domain-Models her. Ähnlich wie bei EMF wird auch bei GMF ein *Generator-Model* verwendet, um die Code-Generierung anzustoßen. Das so erstellte neue `domain`-Projekt benötigt die aus dem ersten Generierungsprozess vorliegenden Sourcen und bildet den gewünschten grafischen Editor.

3.3 XML

Die Ausführung von mit Hilfe der Authoring-Anwendung erstellten Spielszenarien findet auf einem mobilen Endgerät statt. Hierfür ist es notwendig, eine adäquate Technik zum Austausch der benötigten Informationen zu finden.

Auf diesem Gebiet zu einem wichtigen Standard entwickelt hat sich die, vom W3C[2] (World Wide Web Consortium) hervorgebrachte, XML (Extensible Markup Language) [6,27,58]. Diese vor allem im Webbereich oft eingesetzte, flexible Sprache dient der Auszeichnung bzw. der strukturierten, hierarchischen Gliederung von Dokumenten, ist jedoch per se keine Programmiersprache. XML-Dokumente sind herkömmliche Textdateien und daher sowohl für Maschinen als auch für Menschen relativ gut lesbar und einfach zu verstehen. Die Strukturierung der Daten erfolgt mit Hilfe von Elementen, welche jeweils eine Hierarchieebene abbilden. Elemente bestehen aus je einem öffnenden bzw. schließenden Tag (Beschreibung der Informationen) und können sowohl Daten als auch Attribute besitzen. Ein Beispiel dazu zeigt Listing 3.1.

```
1 <lomotain>
2   <task name="Task A" type="question" id="14">
3     <description>Find the item to earn some points!</description>
4     <item name="Item 1" imageFile="Task-A/Image-1.png"/>
5   </task>
6 </lomotain>
```

Listing 3.1: Einfaches XML-Dokument.

Die dargestellten, baumartig strukturierten Daten bestehen aus einem Wurzelelement und drei weiteren Elementen auf verschiedenen Hierarchiestufen. Die Elemente `task` und `item` besitzen neben dem Namen zusätzliche Informationen in Form von Attributen. Diese bestehen wiederum aus einem Namen sowie einem in Anführungszeichen eingeschlossenen Wert. Das Element `description` zeigt, dass Daten nicht nur als Attribute sondern auch direkt zwischen den Element-Tags als Inhalt abgelegt werden können.

XML ist eine erweiterbare Sprache. Dies bedeutet, Entwickler können etwa neue Elemente für ihre speziellen Bedürfnisse erstellen. Dabei müssen jedoch bestimmte, per XML-Spezifikation vorgeschriebene, Bedingungen erfüllt werden. Genügt ein erstelltes XML-Dokument diesen syntaktischen Einschränkungen, so spricht man von Wohlgeformtheit. Die wichtigsten Regeln, die ein wohlgeformtes XML-Dokument einhalten muss, sind:

- Ein XML-Dokument darf nur ein Wurzelelement (äußerstes Element) besitzen.
- Alle Elemente müssen mit Tags gekennzeichnet sein.
- Element- und Attribut-Namen dürfen keine Leerzeichen enthalten.
- Element-Tags dürfen sich nicht überschneiden.

[2] http://www.w3.org

- Attribute eines Elements müssen eindeutige Namen besitzen.

Neben wohlgeformt kann ein XML-Dokument auch gültig sein – dies ist dann der Fall, wenn es einer bestimmten Grammatik (diese definiert u. a. die erlaubten Elemente) entspricht. Grammatiken für XML-Dokumente können mit der eher einfachen DTD (Document Type Definition) oder mit dem mächtigeren XML-Schema erstellt werden.

Verwendung findet die plattformunabhängige XML-Technologie vor allem bei Datenaustausch, Konfigurationsdateien, Webservices, Definition von Benutzerinterfaces, Datenserialisierung etc. In der vorliegenden Arbeit wird XML als internes Datenformat sowie zum Datenaustausch zwischen einer Desktop- und einer mobilen Anwendung eingesetzt.

XML-Verarbeitung

XML-Dokumente können entweder manuell mit einfachen Texteditoren oder programmtechnisch (z. B. mit Java) erzeugt werden. Bei letzterer Variante gibt es wiederum zwei Möglichkeiten: Der Entwickler kann den gewünschten Inhalt (Text) mit Hilfe eines simplen Dateizugriffs im Dateisystem ablegen. Dies ist relativ simpel umsetzbar, effizient und mit jeder Programmiersprache möglich. Jedoch können dabei leicht syntaktische Fehler passieren, womit das Dokument nicht wohlgeformt ist und nicht optimal weiterverwendet werden kann. Diese Fehlerquelle kann mit Hilfe der zweiten Möglichkeit – der Verwendung einer XML-API (Application Programmer Interface) – eliminiert werden. Solche vorgefertigten, für die meisten Programmiersprachen verfügbaren Bibliotheken sind zwar komplexer und weniger effizient, bieten dem Entwickler jedoch entscheidende Vorteile. Dazu gehört z. B. die automatische Validierung des Dokuments (Prüfung auf Wohlgeformtheit).

Nach dem erfolgreichen Generieren und Übertragen der XML-Dokumente müssen diese – wie im vorliegenden Projekt etwa auf einem mobilen Endgerät – zur späteren Verarbeitung eingelesen werden. Diese Aufgabe (das Parsen der Datei) übernimmt üblicherweise eine bestehende Bibliothek. Ähnlich wie bei der Erzeugung von XML-Dokumenten gibt es auch an dieser Stelle mehrere Methoden – ein ereignisbasiertes, ein modellbasiertes und ein pullbasiertes Verfahren.

Ereignisbasierte bzw. SAX-Parser (Simple API for XML) benachrichtigen die Anwendung – etwa beim Lesen eines Start-Tags – in Form von Events. Dies ist zwar sehr effizient und schnell, jedoch relativ komplex. Größter Vorteil dieser Methode im Hinblick auf mobile Endgeräte ist die Tatsache, dass dieses Verfahren relativ wenig Speicher benötigt. Außerdem wird nicht das gesamte Dokument auf einmal eingelesen – es erfolgt eine Zerlegung in kleine Teile, welche sofort verarbeitet werden können. Aufgrund dieser Vorteile eignen sich ereignisbasierte Parser ferner besonders für die Verarbeitung großer Datenmengen.

Modellbasierte Parser – auch DOM-Parser (Document Object Model) genannt – erstellen intern eine hierarchisch aufgebaute Baumstruktur, welche alle im XML-Dokument definierten Elemente abbildet. Damit wird dem Entwickler das komfortable Navigieren im XML-Dokument ermöglicht. Außerdem stehen bei Verwendung dieser Technik immer alle XML-Elemente zur Verfügung, an denen relativ einfach Veränderungen durchgeführt werden können (z. B. neue Elemente anfügen). Modellbasierte Parser lesen die XML-Datei vollständig ein bevor die Verarbeitung beginnt. Dies kann speziell bei großen Dokumenten zu längeren Wartezeiten führen. Ein weiterer, speziell für Mobilgeräte zu berücksichtigender Nachteil dieser Methode ist der hohe Speicherbedarf aufgrund des Aufbaus der internen Baumstruktur.

Pullbasierte Parser besitzen Aspekte der beiden ersten Techniken und stellen die dritte Möglichkeit zum Einlesen von XML-Dokumenten dar. Ähnlich wie bei der ereignisbasierten Methode werden der Anwendung auch hier bereits während des Parsens Daten geliefert. Dies geschieht jedoch nicht in Form von Events, sondern mittels Rückgabeparametern. Ein weiterer Unterschied zur ereignisbasierten Technik, welcher sogleich die Ähnlichkeit mit modellbasierten Parsern herstellt, ist die Tatsache, dass bei der pullbasierten Verarbeitung der Entwickler bzw. die Applikation Anfragen absetzen bzw. Daten anfordern muss und somit die Ablaufkontrolle behält.

Es existieren diverse APIs – etwa JAXP (Java API for XML Processing) – welche mehrere bzw. alle drei XML-Verarbeitungstechniken implementieren. Im Rahmen dieser Arbeit wird der frei verfügbare kXML-Parser eingesetzt – eine XML-API, welche ausschließlich den pullbasierten Ansatz verfolgt. kXML ist aufgrund der geringen Größe sowie des niedrigen Speicherbedarfs eine vor allem für mobile Endgeräte optimale Variante. Neben kXML zum Einlesen wird zur Erzeugung von XML-Dokumenten die modellbasierte Schnittstelle von JAXP verwendet.

3.4 JavaCC

Die ans mobile Endgerät übertragene, auf XML basierende Spielbeschreibung enthält jegliche in der Statechart-Logik definierten Informationen. Hierzu zählen bspw. auch vom Entwickler erstellte Transition-Guards, welche zukünftig u. a. kontextsensitive Ausdrücke beinhalten und so die Anwendung situationsgerecht steuern sollen. Das Parsen bzw. Verarbeiten solcher Eingabedaten (Guards im vorliegenden Projekt) gehört zu einem der wichtigsten Aufgabengebiete der Informatik. Dabei kann es sich um einfache, kommaseparierte Listen oder um komplexe Eingabeströme handeln, welche das Erstellen eines eigenen Parsers erforderlich machen. Solche von Hand geschriebene Programme zum Verarbeiten von Eingabedaten sind oft relativ umfangreich und unübersichtlich, weshalb im Laufe der Zeit diverse Werkzeuge zur automatischen Parser-Generierung entwickelt wurden. Eines dieser Tools ist

JavaCC (Java Compiler Compiler) [13, 52] – ein in Java entwickelter Parser-Generator, welcher unter der BSD-Lizenz[3] (Berkeley Software Distribution) für freie Software steht.

JavaCC besteht aus einem lexikalischen Analysator (Lexer), einem syntaktischen Analysator (Parser) und einem Generator für Verarbeitungsbäume (JJTree). Das Lexer-Modul erstellt aus einer Buchstabenfolge eine Folge von erkannten Token (Symbolen), welche in der Grammatik definiert wurden. Die so erhaltenen Token verwendet der Parser, um die syntaktische Analyse durchzuführen, zusammengehörende Konstrukte zu finden sowie Ausgabeinformationen zu erstellen. Vergleichbar ist dieses Konzept mit dem Satzbau: Der Lexer prüft die Richtigkeit der einzelnen Wörter – der Parser stellt sicher, dass der gesamte Satz korrekt ist. Zur einfachen Weiterverarbeitung der so geprüften Daten wird ein Verarbeitungsbaum erstellt – entweder mittels JJTree oder durch das manuelle Erweitern der generierten Parserklassen.

Abbildung 3.5 visualisiert den üblichen Ablauf beim automatischen Generieren eines Parsers durch JavaCC.

Abbildung 3.5: JavaCC-Workflow (nach [52]).

Dabei wird eine zuvor definierte Grammatik (formale Beschreibung der Sprache) als Eingabeparameter übergeben. JavaCC generiert daraus einen Top-Down-Parser bestehend aus Java-Klassen. Bei der so erhaltenen Anwendung handelt es sich um einen LL(k)-Parser. Dies bedeutet, die Eingabefolge wird von links nach rechts abgearbeitet (Linksableitung) und der Parser kann k Symbole bzw. Token vorausschauen (Lookahead). Bei jeder Ableitungs-Entscheidung wird dieser Lookahead miteinbezogen. Als Verarbeitungsmethode verwendet der von JavaCC generierte Parser den rekursiven Abstieg. Dabei wird jedes Nichtterminalsymbol[4] der Grammatik durch eine – oftmals rekursiv aufgerufene – Funktion abgebildet.

Beispiel-Grammatik

Zur Verdeutlichung des Konzepts soll ein einfaches Beispiel dienen, welches die Addition sowie Subtraktion von beliebig vielen Integer-Werten zulässt (z. B. „14 + 8 – 4“). Listing 3.2 zeigt die dafür notwendige Tokendefinition.

[3] Unter der BSD-Lizenz stehende Software darf für private und kommerzielle Zwecke kopiert, verändert und verbreitet werden. Einzige Bedingung ist das Beibehalten der ursprünglichen Copyright-Informationen.

[4] Nichtterminalsymbole sind im Gegensatz zu Terminalsymbolen Zeichen oder Zeichenfolgen, welche keine definierten Symbole der Grammatik, sondern syntaktische Token darstellen. Diese werden durch die Anwendung von Regeln in Terminalsymbole zerlegt.

```
1 TOKEN: {
2     < INTEGER : (<NUM>)+ >
3   | < ADD : "+" >
4   | < SUB : "-" >
5   | < #NUM: [ "0"-"9" ] > }
```

Listing 3.2: Beispiel-Grammatik – Tokendefinition.

In diesem ersten Abschnitt der Grammatik werden die benötigten Token bzw. Symbole definiert. Im vorliegenden Beispiel werden somit nur Integer- sowie Additions- und Subtraktions-Token erlaubt. Das Symbol `INTEGER` kann dabei beliebig viele Ziffern beinhalten. Ein vor die Token-Definition gestelltes Rautenzeichen (#) stellt sicher, dass das definierte Symbol nur zur Festlegung anderer Symbole verwendet werden darf, jedoch vom Lexer nicht verwendet wird. Neben diesen Token ist es möglich, nicht benötigte Zeichenfolgen (z. B. Leerzeichen) in einem eigens dafür vorgesehenen Abschnitt (`SKIP`) zu definieren – diese werden vom Lexer ignoriert.

Der wichtigste Abschnitt der Grammatik – dargestellt in Listing 3.3 – beinhaltet die Spezifikation der Syntax bzw. die Grammatikregeln. Diese stellen jene Folgen von Token dar, welche gültige Ausdrücke sind.

```
1  TreeNode Statement() : {
2    TreeNode root;
3  } {
4    root = Number() ( root = AddSubExpr(root) )* <EOF>
5    { return root; }
6  }
7
8  TreeNode AddSubExpr(TreeNode _nodeA) : {
9    TreeNode nodeB;
10 } {
11     <ADD> nodeB = Number()
12     { return new TreeNodeAdd(_nodeA, nodeB); }
13   | <SUB> nodeB = Number()
14     { return new TreeNodeSub(_nodeA, nodeB); }
15 }
16
17 TreeNode Number() : {
18   Token t;
19 } {
20   t = <INTEGER>
21   { return new TreeNodeNumber(Integer.parseInt(t.image)); }
22 }
```

Listing 3.3: Beispiel-Grammatik – Regeldefinition.

Es ist ersichtlich, dass die vorliegende Grammatik mit Hilfe von drei Funktionen, welche die Syntax beschreiben, definiert ist. Ausgangspunkt ist dabei die Funktion `Statement()`, welche als Subelemente eine Nummer sowie eine beliebige Anzahl an Additions- bzw. Subtraktions-Ausdrücken erlaubt und die Eingangsdatei bis zum Ende (`<EOF>`) einliest. Die Funktion `AddSubExpr()` teilt sich in zwei Bereiche – je einen für Addition und Subtraktion – und gibt innerhalb dieser die gewünschte Syntax vor. Schließlich bildet die Methode `Number()` einen Integer selbst ab.

Neben den zuvor definierten Token wurden bei der Erstellung dieser Grammatikregeln auch semantische Aktionen zum Prüfen der Syntax bzw. zum Weiterverarbeiten verwendet. In diesem Beispiel wird mit Hilfe der nach der Syntax-Definition angegebenen Aktionen die Generierung einer Baumstruktur verwirklicht: Mit Hilfe einiger vom Entwickler anzulegender Klassen (`TreeNodeAdd`, `TreeNodeNumber` etc.) sowie Rückgabewerten und Übergabeparametern werden Objekte erzeugt, welche sodann zu einer baumartigen Hierarchie zusammengefügt werden. Dadurch ist es zur Laufzeit der Anwendung relativ einfach möglich – etwa durch Verwendung einer geeigneten Interface-Methode – die Knoten der Baumstruktur auszuwerten und zu verarbeiten. Für die Eingabe „14 + 8 – 4“ stellt sich beispielsweise die interne Hierarchie wie folgt dar (Abbildung 3.6):

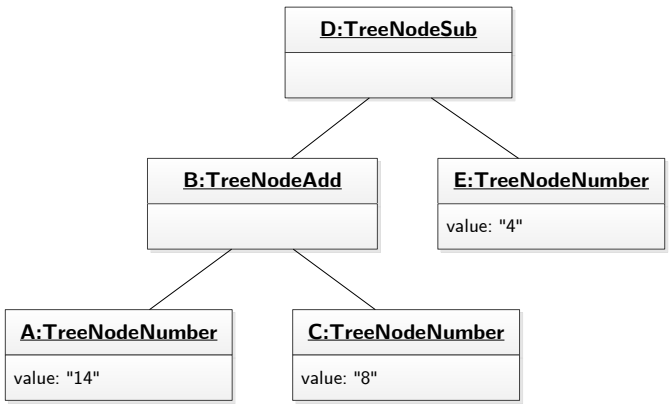


Abbildung 3.6: Intern erzeugter Verarbeitungsbaum.

Für jedes erkannte Token des Eingabestroms wird dabei ein Objekt erzeugt, welche hierarchisch zu einem Verarbeitungsbaum verbunden werden. Durch das Erweitern der in der Baumstruktur verwendeten Klassen können die Informationen weiterverarbeitet und Funktionalitäten wie etwa das Addieren bzw. Subtrahieren der einzelnen Integer-Werte realisiert werden.

Kapitel 4

Umsetzung

4.1 Lomotain

Der in der vorliegenden Arbeit beschriebene und auf Statecharts basierende Ansatz zur Integration von Kontextsensitivität in Anwendungen wurde als Erweiterung zum bestehenden Projekt Lomotain [22, 48] entwickelt. Dieser Abschnitt soll nun die Grundidee dieses Projekts vermitteln, eine abstrakte Systemarchitektur darstellen sowie alle notwendigen Informationen über Lomotain beschreiben, welche in weiterer Folge wichtig für die Umsetzung des erarbeiteten Konzepts sind.

4.1.1 Grundidee

Lomotain steht für **Lo**cation Based **Mo**bile Gaming and Enter**tain**ment und wurde 2007 von der FH Oberösterreich[1] – Studiengang Mobile Computing und zwei Partnerunternehmen (FAW[2], MOWIS[3]) als Forschungsprojekt ins Leben gerufen. Ziel des Projekts ist es, Tourismusregionen und Themenparks mit Hilfe neuartiger, mobiler, spielbasierter Adventure-Guides basierend auf modernen Technologien interessanter und attraktiver für Besucher zu gestalten. Lomotain unterscheidet sich dabei klar von üblichen Tourismusführern, welche meist große Aufmerksamkeit vom Benutzer verlangen. Die hierbei verfolgte Strategie lässt den Anwender weniger mit dem Endgerät, sondern vielmehr mit der Umwelt bzw. Objekten daraus interagieren. Somit rückt die Anwendung auf dem Mobilgerät etwas in den Hintergrund und Besucher können ihre Aufmerksamkeit auf die eigentlichen Interessen – das besuchte Gebiet bzw. die Umwelt – richten. Neben den realen Objekten (z. B. Bäume) können vor allem für spielbasierte Szenarien auch fiktive Gegenstände (z. B. Schatztruhen) zur Interaktion geschaffen werden, welche – verknüpft mit Objekten aus der Realität – die Ausführung des Abenteuers ermöglichen.

[1] http://www.fh-ooe.at
[2] http://www.faw.at
[3] http://www.mowis.at

Etwaige Einsatzbereiche des Systems sind in erster Linie Themenparks, welche bspw. durch das Anbieten bisher nicht existenter Attraktivitäten Aufmerksamkeit auf sich lenken und dadurch höhere Besucherzahlen erreichen möchten. Potenzielle, mit Lomotain umsetzbare, Attraktionen für solche Organisationen oder Tourismusgebiete sind etwa Spielszenarien zur unterhaltsamen Erforschung des Gebiets anhand von Rätselaufgaben und Fragestellungen. Neben Unternehmen können auch ganze Städte bzw. Regionen von Lomotain profitieren – z. B. durch das Bereitstellen von interaktiven Natur- oder Städteführern.

Als Interaktionsmechanismus wird von Lomotain RFID (Radio Frequency Identification) bzw. NFC (Near Field Communication) verwendet. Geeignete Tags inklusive etwaiger Beschreibung werden von Parkbetreibern bzw. Tourismusorganisationen an den für die Spielszenarien benötigten Objekten angebracht. Mittels passender Mobilgeräte sind Benutzer somit in der Lage, RFID-Tags auszulesen und die dahinter liegenden Aktionen auszuführen bzw. das Abenteuer zu steuern. Vorteile solcher Tags, welche jeweils eine eindeutige ID besitzen, sind sowohl Robustheit gegenüber Wind und Wetter als auch Vandalismussicherheit, eine geeignete Anbringung (z. B. hinter Kunststoffglas) vorausgesetzt. Darüber hinaus sind diese im einstelligen Euro-Bereich erhältlichen Chips bei Bedarf für mehrere Spielszenarien gleichzeitig einsetzbar, woraus sich für Parkbetreiber eine Kosten- sowie Arbeitsminimierung ergibt.

Um überhaupt Anwendungsszenarien für Lomotain erstellen zu können, sind einige Basisaufgabentypen notwendig. Diese sind bereits ins System integriert und können von Parkbetreibern verwendet werden, um ein größeres Spielkonzept zu erstellen:

- **Multiple-Choice-Fragen** fordern den Benutzer auf, aus einer Menge an Antwortmöglichkeiten die richtige zu finden. Dabei müssen die Antwortmöglichkeiten in Form von zugehörigen RFID-Tags erst gefunden bzw. aktiviert werden, um zu erfahren, was sich dahinter verbirgt. Wurden alle Tags ausgelesen, kann der Benutzer seine Antwort durch zweimaliges Einlesen des RFID-Tags – analog zum hinreichend bekannten Doppelklick – auswählen.

- Das **Sammeln von Gegenständen** funktioniert ähnlich wie der bereits beschriebene Aufgabentyp. Dabei müssen vom Anwender lediglich bestimmte, meist an Objekten angebrachte, RFID-Tags gefunden sowie ausgelesen werden. Das System registriert die gefundenen Tags und visualisiert dem Benutzer, welche Objekte bereits entdeckt wurden sowie die Gesamtanzahl der zu findenden Gegenstände. Lomotain unterstützt zwei verschiedene Modi dieses Aufgabentyps – einerseits das simple Finden aller Objekte, andererseits auch das Erfassen von Gegenständen in einer bestimmten, vordefinierten Reihenfolge.

- Beim **Labyrinth** stellt sich dem Benutzer die Aufgabe, eine Spielfigur durch ein solches zu führen. Jede Richtung (links, rechts, oben, unten) wird durch jeweils einen RFID-Tag abgebildet, mit dessen Hilfe die Figur gesteuert werden kann. Das System stoppt die Bewegung automatisch bei Wegkreuzungen und Hindernissen. Erreicht die Spielfigur durch adäquate Navigationskommandos des Benutzers das Ziel, ist die Aufgabe gelöst.

Neben RFID stützt sich Lomotain auf einer Reihe weiterer Technologien – die wichtigsten davon sind Java ME (Java Micro Edition), Statecharts und XML (Extensible Markup Language). Zentrale Einheit des Systems ist eine flexible Logik, welche mit Hilfe von Zustandsautomaten definiert wird und die Basis eines jeden Lomotain-Spiels bildet. Darüber hinaus hat dieser Ansatz den Vorteil, dass einzelne Spiele – die Lomotain-Anwendung vorausgesetzt – individuell auf Mobilgeräten installiert werden können. Somit können Anwender, welche die Basisanwendung bereits besitzen, die Applikation durch einfaches Aufspielen neuer Szenarien (z. B. vor Ort in einem Themenpark per Bluetooth) problemlos selbständig erweitern.

4.1.2 Systemarchitektur

Um die soeben vorgestellten Ideen umsetzen zu können, werden diverse Komponenten benötigt, welche gemeinsam das Lomotain-System verkörpern. Die wichtigsten Module dieses Systems zeigt Abbildung 4.1.

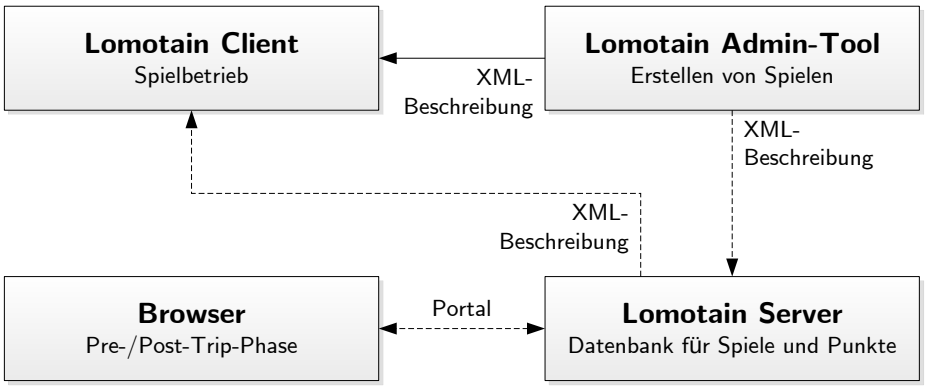


Abbildung 4.1: Überblick der Lomotain-Systemarchitektur.

Dabei gibt es einerseits das Lomotain Admin-Tool, welches die Erstellung von Spielszenarien, die Definition der Anwendungslogik sowie das Einpflegen von Inhalten (Bilder, Musik, Text etc.) ermöglicht. Diese Desktop-Anwendung ist primär für Parkbetreiber und Tourismusorganisationen gedacht, welche ihren Besuchern Lomotain-Abenteuer zur Verfügung stellen möchten.

Die aus dem Admin-Tool generierte Spielbeschreibung besteht aus einem XML-Dokument sowie allen benötigten Bild- bzw. Tondateien. Diese Beschreibung wird derzeit direkt ans Mobilgerät des Besuchers (z. B. per USB oder Bluetooth) übertragen. Der dort installierte Lomotain Client – die Mobilanwendung des Systems – liest alle vorhandenen Spielbeschreibungen ein und ermöglicht dem Benutzer somit das Starten bzw. Ausführen aller verfügbarer Abenteuer.

Neben Admin-Tool und Client-Anwendung zeigt das dargestellte Blockschaltbild zwei vorgesehene und entwickelte, jedoch aktuell nicht verwendete Komponenten. Dazu gehört der Lomotain Server, welcher eine Datenbank inkludiert und für das persistente Abspeichern bzw. Weiterreichen von Spielbeschreibungen an Anwender sowie zur Vorhaltung von Punkteständen und Spielstati konzeptioniert wurde. Vor allem in Hinblick auf einen Mehrspielermodus und Highscore-Listen ist solch ein Modul erforderlich. Auch ein vorgesehenes Webportal, für Benutzer per Browser erreichbar, benötigt die am Server abgelegten Informationen – etwa zur Visualisierung von Punkteständen und zur Veröffentlichung neuer Lomotain-Abenteuer.

Die von Lomotain gewählten Stationen, welche Spielszenarien im Laufe ihrer Entwicklung durchschreiten, sind in Abbildung 4.2 dargestellt.

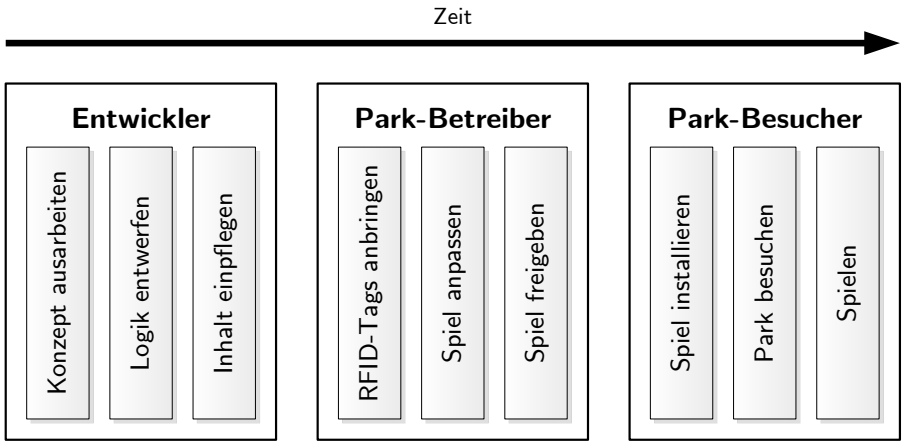


Abbildung 4.2: Lomotain-Lifecycle (nach [22]).

Entwickler der Lomotain-Plattform oder Organisationen, welche einzelnen Themenparks bzw. Regionen den Einstieg in dieses System erleichtern wollen, beginnen mit der Ausarbeitung von Spielkonzepten, der Definition dazu passender Anwendungslogik und dem Einspeisen von generell gültigen Inhalten. Das dabei behilfliche Lomotain Admin-Tool benötigen ebenso Parkbetreiber, um das zuvor von Entwicklern erstellte Spiel mit für die lokalen Gegebenheiten wichtigen Informationen zu vervollständigen. Hierzu gehören etwa die in der Region installierten RFID-Tags sowie etwaige

Brandinginformationen, um das Spielszenario mit dem jeweiligen Park in Verbindung zu bringen. Nach dem Veröffentlichen des Spiels ist die letzte, jedoch wichtigste Station der Parkbesucher, welcher die Spielbeschreibung auf das Endgerät überträgt und im jeweiligen Tourismusgebiet das geeignete Abenteuer startet.

4.1.3 Erweiterungen

Dieses soeben beschriebene erste Lomotain-System bildet die Grundlage einiger möglicher Erweiterungen bzw. Folgeprojekte. Dazu zählt etwa die, aufgrund der relativ geringen Verbreitung der NFC-Technologie auf mobilen Endgeräten notwendige, Integration alternativer Steuerungsmechanismen. Eine Reihe solcher Interaktionstechnologien – z. B. 2D-Code, Gestenerkennung, Tasteneingabe – wurde bereits für Lomotain angedacht und teilweise auch – im Rahmen von Vorprojekten dieser Arbeit – ins System integriert.

Die in einer Diplomarbeit aufgegriffene Erweiterung des Systems behandelt die automatische Anpassung der Anwendung an das jeweilige Mobilgerät bzw. dessen Ressourcen. Beachtet wird dabei auch, dass die Anwendung auf möglichst vielen verschiedenen mobilen Endgeräten ausgeführt werden kann. Somit ist es nicht notwendig, Parkbesuchern eigens dafür vorgesehene Endgeräte zur Verfügung zu stellen – die Benutzer können die Anwendung auf den eigenen Mobiltelefonen installieren und verwenden.

Nicht als Erweiterung, jedoch als mögliche Ergänzung zum aktuellen Projekt angedacht ist ein praxisbezogener Feldtest. Dabei bietet sich das Gebiet des kooperierenden Naturparks Agrarium[4] an, welcher das Lomotain-System zukünftig als eine innovative Attraktion verwenden könnte. In einem ersten Schritt sollen Parkbesucher die Möglichkeit erhalten, die Anwendung mittels einiger Test-Szenarien bzw. Basisaufgaben ausführlich zu testen. Neben dem betriebswirtschaftlichen Mehrwert für den Park bietet ein solcher Test auch die Möglichkeit, Erfahrungsberichte von Anwendern einholen, auswerten und in die weitere Systementwicklung einfließen lassen zu können.

Die bereits in Abschnitt 1.3 erwähnte, im Rahmen der vorliegenden Diplomarbeit zum Ziel gesetzte, Konzeptionierung und Umsetzung eines auf Statecharts basierenden Ansatzes zur kontextsensitiven Anwendungsadaption stellt ebenfalls eine Erweiterung des bestehenden Systems dar. Damit ist es später möglich, alle Benutzer – z. B. abhängig von ihrem jeweiligen Alter – individuell zu unterstützen bzw. Rätselaufgaben passend zur richtigen Alterskategorie bereitzustellen.

In den folgenden Abschnitten wird dieser neue Ansatz zur kontextsensitiven Anwendungsadaption näher erläutert. Dabei wird – ausgehend von identifizierten Anforderungen – ein Konzept (u. a. im Hinblick auf benötigte Kontextparameter sowie der Integration kontextsensitiver Aspekte in die

[4] http://www.agrarium.at

Statechart-Logik) erstellt, welches als Grundlage der späteren Implementierung dient. Deren relevante Aspekte werden dabei – gemeinsam mit der detaillierten Systemarchitektur – ebenso ausführlich erläutert wie eine den Abschluss dieses Kapitels bildende Evaluierung des umgesetzten Konzepts.

4.2 Integration Kontextsensitivität

4.2.1 Anforderungen

Vom primären Ziel der vorliegenden Arbeit – der Umsetzung kontextsensitiver Anwendungsadaptionen auf Basis von Statecharts – lassen sich einige Anforderungen ableiten, welche an das zu entwickelnde System bzw. dessen Teilkomponenten gestellt und in diesem Abschnitt erläutert werden. Wichtigster Aspekt dabei ist die Kontextsensitivität selbst – mit Lomotain erstellten Anwendungen bzw. Spielszenarien muss es möglich sein, sich an die aktuell herrschende Kontextsituation bzw. Umgebung anzupassen sowie darauf zu reagieren. Dabei sollen einerseits Inhaltsobjekte (Text, Bilder, Melodien etc.) kontextsensitiv adaptiert werden können – z. B. um gleiche Spielszenarien für verschiedene Altersgruppen mit unterschiedlichen Schwierigkeitsstufen erstellen zu können. Andererseits soll der Ansatz ebenfalls die kontextsensitive Adaption einer auf Statecharts basierenden Anwendungslogik (z. B. die des Lomotain-Projekts) erlauben. Besonderer Anreiz dabei ist die Tatsache, dass eine solche – auf Zustandsautomaten beruhende – Möglichkeit der kontextbasierten Anwendungsadaption bisweilen in der Praxis nicht existiert (siehe Abschnitt 2.3.2). Der im Rahmen dieser Diplomarbeit vorgestellte Ansatz soll jedoch zeigen, dass es auch damit möglich ist, Anwendungen kontextsensitiv zu gestalten.

Ebenso wichtig wie der kontextsensitive Aspekt selbst sind die dafür benötigten Kontextparameter bzw. Kontextinformationen. Es gibt eine Unmenge solcher Daten der Umgebung, welche die Grundlage für kontextbezogene Anwendungen bilden. Für das vorliegende Projekt müssen im Rahmen der Konzepterstellung die geeignetsten und nützlichsten Kontextparameter identifiziert werden, um diese im späteren Verlauf optimal einsetzen zu können. Bei falscher Wahl dieser Basisinformationen ist es u. U. möglich, dass einige davon für Lomotain unbrauchbar sind und erstellte Lomotain-Spielszenarien nur unzureichend kontextsensitiv gestaltet werden können.

Eine weitere Anforderung ergibt sich aus den benötigten Kontextdaten. Um diese Informationen einerseits einlesen, andererseits gesammelt an die betreffenden Module weiterreichen zu können, wird u. a. ein Mechanismus zur Kontext-Akquisition benötigt. Dieser sammelt die für die Anwendung wichtigen Kontextinformationen (z. B. mittels Sensoren, Benutzerprofil etc.) und legt diese mit Hilfe eines geeigneten Datenmodells im System ab. Die Informationen sollen im weiteren Verlauf von einem zentralen Kontext-Distributor verwaltet und an die interessierenden Komponenten weitergegeben werden.

Ein für die Anpassung der Applikation an die äußeren Gegebenheiten unabdingbarer Bestandteil sind sogenannte Kontext-Regeln. Diese legen die Reaktion bzw. Adaption des Systems bezogen auf die aktuelle Kontextsituation fest. Wichtige Anforderung an das System ist deshalb das Bereitstellen einer Möglichkeit zur Integration verfügbarer Kontextparameter in solche Regelsätze. Eine vom System vorgegebene Syntax ist des Weiteren notwendig, um die korrekte Struktur der Regeln vorzugeben. Dazu soll eine im Lomotain-System bereits existierende Grammatik für Bedingungen von Statechart-Transitions weiterentwickelt werden. Neben der Erweiterung dieser Regelsyntax ist es ebenfalls notwendig, die bereits existierende Statechart-Logik um kontextsensitive Konzepte zu ergänzen, um die zukünftigen Aufgaben des Entwicklers (dargestellt in Abbildung 4.3) zu ermöglichen.

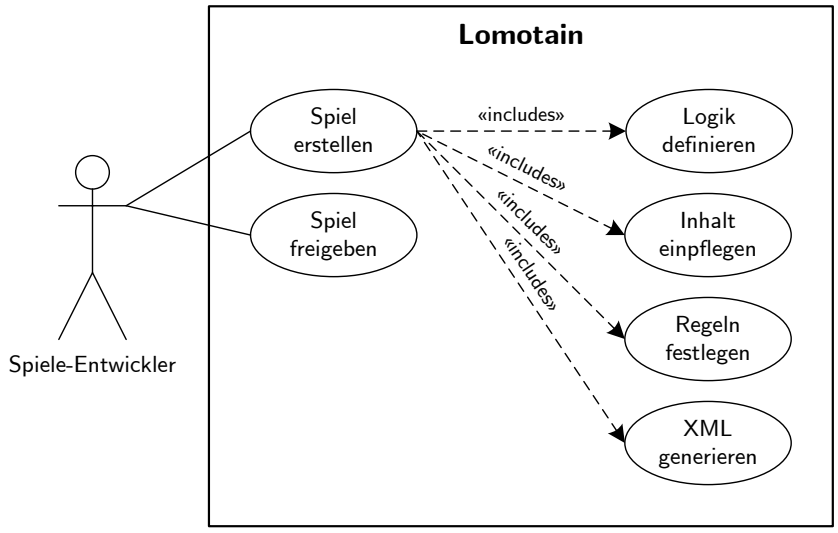


Abbildung 4.3: Use-Case 1 – Aufgaben des Entwicklers.

Dabei ersichtlich sind diverse Aufgaben, welche ins Arbeitsumfeld des Spiele-Entwicklers bzw. Parkbetreibers fallen – z. B. das Erstellen von Spielszenarien. Diese Aufgabe wird mit Hilfe des Admin-Tools abgearbeitet und inkludiert die Definition der Anwendungslogik, das Erfassen von Inhalten sowie das Festlegen von kontextbezogenen Regeln. Das Generieren eines XML-Dokuments, welches das definierte Spielszenario beschreibt, ist ebenso Teilaufgabe des Entwicklers. Dadurch ergibt sich, bezogen auf die Integration des neuen Adaptionsansatzes ins Lomotain-System, eine weitere Anforderung – nämlich das Erweitern dieser auf XML basierenden Schnittstelle um kontextbezogene Informationen. Erst damit ist es einerseits dem Admin-Tool möglich, diese Aspekte in die Spielbeschreibung zu integrieren. Andererseits wird die Client-Anwendung befähigt, diese Daten einzulesen und im Verlauf der Anwendungsausführung zu nutzen.

Neben den Entwicklern stellen Parkbesucher bzw. Anwender einen weiteren wichtigen Personenkreis im Rahmen des Lomotain-Systems dar. Abbildung 4.4 zeigt die wichtigsten Aufgabenstellungen dieser Personen.

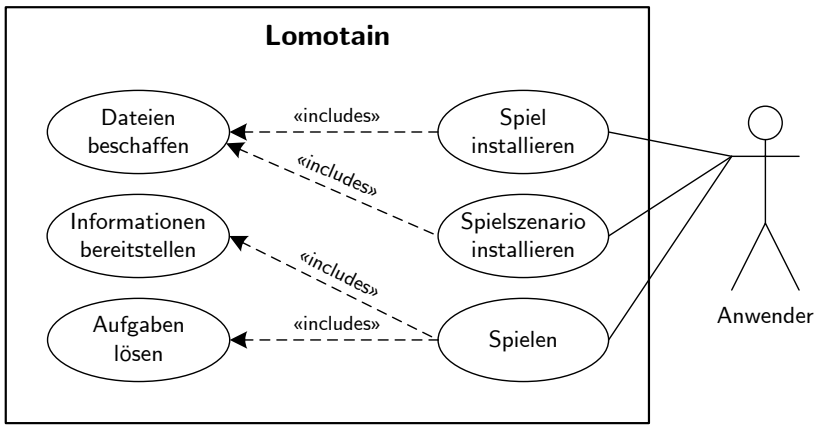


Abbildung 4.4: Use-Case 2 – Aufgaben des Anwenders.

Die letzte bedeutende und zugleich einzige Anforderung, welche sich aus diesem zweiten Use-Case ergibt, ist das Bereitstellen einer Möglichkeit zur Erfassung der für das System wichtigen Kontextinformationen. Alle weiteren Teilbereiche der Use-Cases werden bereits durch das ursprüngliche System abgedeckt.

4.2.2 Benötigte Kontextparameter

Um die benötigten Kontext-Regeln aufstellen zu können, werden Kontextparameter benötigt, welche zur Laufzeit den aktuellen Werten (z. B. gegeben durch die Einstellungen im Benutzerprofil) entsprechen. Da es unzählige Kontextparameter gibt und die Unterstützung aller im Rahmen dieser Arbeit nicht sinnvoll ist, werden im Folgenden die ausgewählten Parameter mitsamt Begründung dargelegt. In diesem Zusammenhang soll auch auf Abschnitt 2.3.2 verwiesen werden, welcher die am häufigsten eingesetzten Kontextparameter anderer Projekte auflistet.

- **Alter:** Um zwischen verschiedenen Altersgruppen unterscheiden sowie die Inhalte der Spielszenarien dementsprechend anpassen zu können, wird das exakte Alter des aktuellen Benutzers benötigt.
- **Benutzerinteressen:** Da jeder Anwender andere Interessen hat, könnte mit diesem Kontextparameter die Auswahl zwischen verschiedenen Spielszenarien getroffen werden. So könnte etwa für jede Interessensgruppe eine angepasste Version einer Aufgabe zur Verfügung stehen und je nach Benutzerinteresse die passende davon gestartet werden.

- **Geschlecht:** Um etwa für Mädchen und Buben unterschiedliche – an den Zielgruppen orientierte – Aufgabenstellungen bzw. Inhalte anbieten zu können, ist es notwendig, diesen Kontextparameter ins System aufzunehmen.

- **Tageszeit:** Die aktuelle Tageszeit spielt vorwiegend bei Aktivitäten im Freien eine wichtige Rolle – etwa ist es eventuell nicht möglich, bestimmte Spielszenarien bei Nacht (Dunkelheit) durchzuführen. Ebenfalls denkbar wäre es, den gesamten Spielverlauf an die aktuelle Tageszeit anzupassen.

- **Temperatur:** Die aktuelle Temperatur könnte ebenfalls für Aktionen in der freien Natur interessant sein. Einerseits kann damit sichergestellt werden, dass bei sehr extremen Minusgraden nur geeignete Aufgaben im Außenbereich durchgeführt werden. Andererseits können sich aus der aktuellen Temperatur auch interessante Fragestellungen eröffnen (z. B. Gefrierpunkt des Wassers).

- **Wetter:** Neben Temperatur und Uhrzeit hat auch das Wetter Einfluss auf Aktivitäten im Freien – etwa ist es nicht gewünscht, bei Regen eine Wanderung auf schlechtem Untergrund durchführen zu müssen.

Nicht berücksichtigt wurde in diesem Zusammenhang der am häufigsten genutzte Kontextparameter – die Position. Aufgrund der Tatsache, dass es für Lomotain-Spiele angedacht ist, Areale zu definieren, in denen das jeweilige Spielszenario abgewickelt werden kann, ist hier bereits indirekt die Verwendung dieser Kontextinformation gegeben.

Weitere oft verwendete Kontextinformationen sind Eigenschaften der verwendeten Hardware – in diesem Fall Informationen über das Mobiltelefon. Einige davon werden – zum automatischen Anpassen der Anwendung an die Eigenschaften des mobilen Endgeräts – ebenfalls bereits von Lomotain verwendet und deshalb im Rahmen dieser Arbeit nicht näher behandelt.

Die Erfassung der sechs gewählten Kontextinformationen ist nicht primärer Gegenstand des Projektes bzw. der Diplomarbeit. Deshalb soll eine relativ einfache und häufig verwendete Methode eingesetzt werden – das Angeben von Benutzerinformationen über ein Profil. Vor dem Start eines Lomotain-Abenteuers soll der Benutzer dadurch die Möglichkeit haben, sein eigenes Benutzerprofil abzurufen sowie Informationen darin zu ändern. Zur Vereinfachung der Eingaben könnten vordefinierte Werte als Auswahlmöglichkeiten angeboten werden – der Benutzer muss sodann lediglich das passendste Attribut wählen.

Das Ablegen der gesammelten Daten soll mit Hilfe einer einfachen Modellierungsart – sogenannten Key-Value-Paaren – erfolgen. Dabei besitzt jedes Kontextattribut genau einen aktuellen Wert ohne weitere Metainformationen. Eine derartige Modellierungsart ist für das vorliegende Projekt voll-

kommen ausreichend – ferner ist das Thema der Kontext-Modellierung nicht vorrangiger Inhalt dieser Arbeit.

Um die Kontextinformationen anderen Systemkomponenten zur weiteren Verarbeitung zur Verfügung stellen zu können, ist ein Distributionsmechanismus notwendig. Dabei soll der Singleton-Ansatz [58] genutzt werden, welcher sicherstellt, dass von einem Objekt (z. B. einer Java-Klasse) nur eine Instanz existiert. Dadurch kann von jedem beliebigen Applikationsmodul auf dieses Singleton-Objekt zugegriffen und mit der selben Datenbasis – in diesem Fall Kontextinformationen – gearbeitet werden.

4.2.3 Erweiterung der Statechart-Logik

Die soeben identifizierten Kontextparameter bilden die Basis der kontextsensitiven Anpassungen innerhalb des Lomotain-Systems. Da die gesamte Logik aller Spielszenarien (z. B. Multiple-Choice-Frage) auf Statecharts basiert, müssen auch die situationsbedingten Aspekte in dieses Konzept einfließen. Im Folgenden werden einige Möglichkeiten vorgestellt, welche die Erweiterung der auf Statecharts basierenden Anwendungslogik hinsichtlich Kontextsensitivität erlauben.

Ansatz 1 – Kontext-Bedingungen

Eine Möglichkeit, Kontextsensitivität in Statecharts einzubinden, ist das Hinzufügen neuer Transition-Guards, welche eine Referenz auf aktuelle Kontextparameter beinhalten (z. B. Alter). Bisher definierte Statechart-Logiken müssen dabei um die gewünschten Kontextsituationen erweitert werden, welche sich durch verschiedene Zweige im Diagramm abbilden. Diese Zweige können unterschiedliche Aktionen ausführen sowie diverse unterschiedliche Inhalte einbinden und sind in Abbildung 4.5 dargestellt.

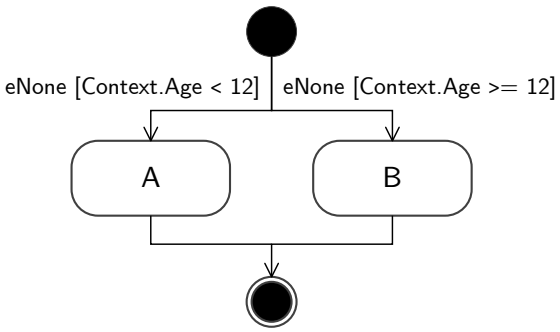


Abbildung 4.5: Ansatz 1 – Kontext-Bedingungen.

Der flexible, einfache Ansatz ermöglicht sowohl die kontextsensitive Anpassung der Anwendungslogik als auch die situationsabhängige Adaption

darzustellender Informationen und ist außerdem hinsichtlich verwendeter Kontextparameter beliebig erweiterbar. Des Weiteren sind keine Änderungen am bestehenden Statechart-Editor notwendig und pro Bedingung mehrere Kontextparameter nutzbar. Hingegen bedeutet dieser Ansatz bei Nutzung vieler Kontextsituationen relativ umfangreiche sowie schwer lesbare Diagramme, da sich die wichtigen Elemente in den Transition-Guards verstecken. Gleich bleibende Logik führt außerdem zu einem gewissen Maß an Redundanz, da gesamte Zweige kopiert werden müssen.

Ansatz 2 – Kontext-States

Ähnlich wie Ansatz 1 benutzt auch der zweite Transitions und Guards, um auf bestimmte Kontextsituationen reagieren zu können. Dabei wird jedoch ein eigens dafür vorgesehener Kontext-State verwendet, welcher den gewünschten Kontextparameter darstellt. Ausgehend von diesem Knoten – illustriert in Abbildung 4.6 – werden wiederum für jede gewünschte Kontextsituation Transitions sowie weiterführende Zweige erstellt.

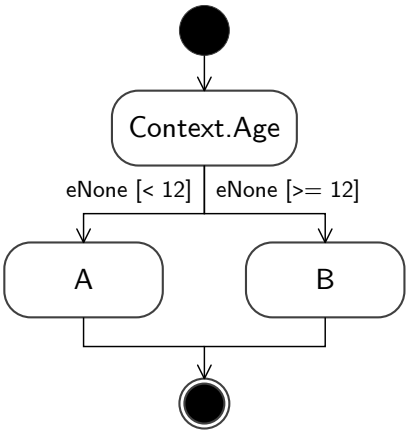


Abbildung 4.6: Ansatz 2 – Kontext-States.

Bei Verwendung dieses relativ einfachen und verständlichen Verfahrens muss der bestehende Statechart-Editor um neue Objekte (Kontext-States) erweitert werden – alle anderen benötigten Mechanismen sind bereits vorhanden. Ebenso wie Ansatz 1 ermöglicht auch dieser die kontextsensitive Anpassung von Inhalt und Logik, jedoch nur mit Hilfe eines einzigen Kontextparameters je Bedingung. Ebenso neigt dieser Ansatz zu Redundanz und Komplexität, ist jedoch hinsichtlich verwendeter Kontextinformationen beliebig erweiterbar und erhöht durch den Einsatz eigener States die Lesbarkeit des Diagramms.

Ansatz 3 – Sub-Regionen

Ein weiteres Verfahren, um kontextbezogene Informationen in Statecharts zu berücksichtigen, ist das Verwenden von Sub-Regionen. Abbildung 4.7 visualisiert diesen Ansatz.

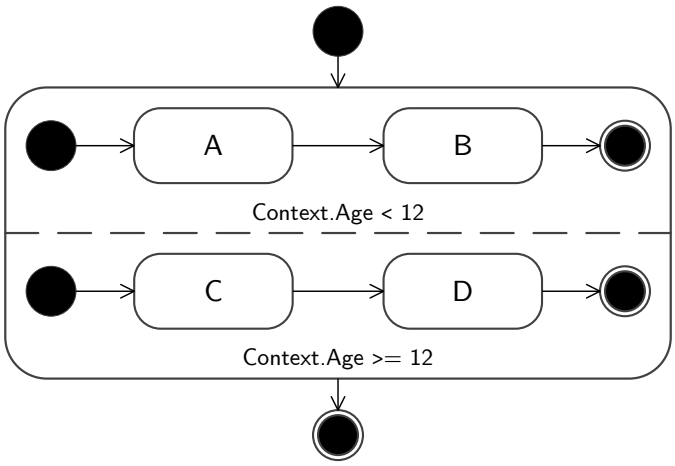


Abbildung 4.7: Ansatz 3 – Sub-Regionen.

Dabei wird ein State in verschiedene Teilbereiche gegliedert, welche jeweils mit einer kontextbezogenen Bedingung versehen werden. Zur Laufzeit wird der aktuell gültige Bereich aktiviert und dessen interne Logik für das gerade ausgeführte Spiel verwendet.

Da das Konzept der Sub-Regionen im aktuellen Statechart-Editor nicht vorgesehen bzw. implementiert ist, erfordert der Einsatz dieser Methode eine Erweiterung des Admin-Tools. Das einfache, verständliche Verfahren ermöglicht die kontextsensitive Adaption von Logik sowie Inhalten und ist hinsichtlich verwendeter Kontextparameter beliebig erweiterbar. Das resultierende Diagramm spiegelt zwar das Verhalten wider, beinhaltet jedoch u. U. Redundanz und hohe Komplexität durch gleich bleibende Logik bzw. umfangreiche Sub-Regionen.

Ansatz 4 – Kontextsensitive Inhalte

Eine weitere Möglichkeit, um auf wechselnde Situationen zu reagieren, ist die vollständige Abkapselung der kontextsensitiven Adaption von der Statechart-Logik – dargestellt in Abbildung 4.8.

Da im aktuellen Konzept viele Elemente (Text, Bilder, Musik) durch sogenannte Content-Objekte und zugehörige Methodenaufrufe eingebunden werden, könnten diese um kontextsensitive Bedingungen erweitert werden. Eine Möglichkeit wäre es, für jedes Content-Objekt mehrere Inhalte mitsamt kontextbezogenen Bedingungen zu definieren (z. B. verschiedene Bilder

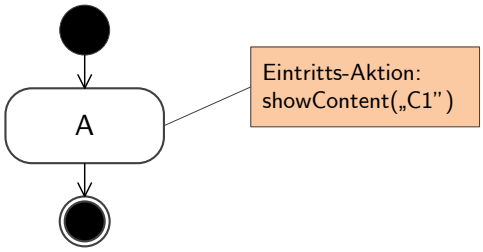


Abbildung 4.8: Ansatz 4 – Kontextsensitive Inhalte.

mit verschiedenen Fragestellungen). Zur Laufzeit der Anwendung muss somit einzig jene Inhaltsdefinition des jeweiligen Content-Objekts gewählt werden, dessen Kontext-Bedingung aktuell gültig ist. Die verschiedenen Content-Objekte werden dabei – wie in Tabelle 4.1 ersichtlich – wiederum mit Hilfe von Kontextparametern und darauf basierenden Bedingungen definiert.

Content-Objekt	Bedingung	Inhalt
C1	`Context.Age < 12`	Aufgabe1a.png
C1	`Context.Age >= 12`	Aufgabe1b.png
C2		Aufgabe2.png

Tabelle 4.1: Definition der Content-Objekte.

Wichtig dabei ist, dass es in jedem Fall eine gültige Bedingung geben muss – ansonsten werden dem Benutzer u. U. keine Inhalte präsentiert. Erleichtert werden soll dieser Umstand durch das Angeben von sogenannten Default-Situationen, wobei die ansonsten genutzte Bedingung leer bleibt. Dieser Mechanismus erleichtert außerdem das Erstellen von Szenarien, welche keine kontextbezogenen Aspekte berücksichtigen. Der zum bestehenden Statechart-Editor kompatible Ansatz 4 ändert des Weiteren die Diagramm-Komplexität nicht und verursacht keine Redundanz durch kopierte Zweige. Jedoch ist damit die Logik nicht kontextsensitiv anpassbar, Erweiterungen im Admin-Tool sind nötig und das erzeugte Diagramm lässt nicht direkt erkennen, wann dem Benutzer welcher Inhalt visualisiert wird.

Ansatz 5 – Kombinierter Ansatz

Den diversen Nachteilen der soeben vorgestellten Ansätze könnte die Kombination zweier gefundener Konzepte entgegenwirken. Dabei bietet sich etwa die Verknüpfung von Ansatz 1 (Kontext-Bedingungen) mit Ansatz 4 (Kontextsensitive Inhalte) an. Damit ist es möglich, simple Content-Änderungen (z. B. verschiedene Aufgabenstellungen) mit Ansatz 4 abzudecken, während

umfangreiche – in die Logik eingreifende – Kontext-Adaptionen durch Ansatz 1 abgebildet werden. Somit würde man – vorausgesetzt der jeweilige Ansatz wird für den korrekten Teilbereich verwendet – die Vorteile beider nutzen bzw. einige Nachteile (z. B. Diagramm-Komplexität, Redundanz) eliminieren. Bestehen bleibt jedoch die erforderliche Erweiterung der Content-Verwaltung im Admin-Tool. Dies ist angesichts der zu erwartenden Ergebnisse bei Verwendung der kombinierten Methode aber durchaus zu erdulden.

Um die fünf soeben vorgestellten Ansätze optimal vergleichen zu können, zeigt Tabelle 4.2 eine Bewertung der wichtigsten Kriterien dieser Konzepte. Vor- bzw. Nachteile im Hinblick auf die einzelnen Merkmale werden dabei durch Plus- (+) bzw. Minus-Symbole (−) dargestellt.

	Ansatz				
Merkmal	**1**	**2**	**3**	**4**	**5**
Kontextsensitive Logik/Inhalte	+/+	+/+	+/+	−/+	+/+
Kompatibel zum Statechart-Editor	+	−	−	+	+
Kompatibel zur Content-Verwaltung	+	+	+	−	−
Mehrere Parameter pro Bedingung	+	−	+	+	+
Diagramm-Komplexität	−	−	−	+	+
Redundanz	−	−	−	+	+

Tabelle 4.2: Bewertung der Ansätze zur Erweiterung der Statechart-Logik.

Aus den dargestellten Vor- und Nachteilen der fünf verschiedenen Methoden kristallisierte sich der kombinierte Ansatz (Ansatz 5) als der passendste heraus. Grund dafür ist einerseits die Eliminierung diverser Nachteile, welche sich aus der Verwendung einzelner Konzepte ergeben. Andererseits deckt dieses Verfahren alle für Lomotain benötigten Anforderungen ab – wichtigste dabei ist die Möglichkeit, sowohl Inhalte als auch Anwendungslogik kontextsensitiv adaptieren zu können. Deshalb verwendet die in Abschnitt 4.4 beschriebene Implementierung eines auf Statecharts beruhenden Konzepts zur kontextsensitiven Anwendungsadaption diesen kombinierten Ansatz.

4.2.4 Erweiterung der Regelsyntax

Ein weiterer wichtiger Aspekt im Rahmen der Konzepterstellung ist die Erweiterung der bestehenden Regelsyntax. Die darüber definierten Bedingungen in Form von Transition-Guards werden im Lomotain-System bzw. Statechart-Editor verwendet, um die gewünschte Anwendungslogik abzubilden. Die exakte in JavaCC formulierte Grammatik dieser ursprünglichen Regeln beschreibt Scheuchenegger [48] – wichtig dabei ist jedoch, dass kei-

nerlei Kontextparameter miteinbezogen werden können. Bereits integriert ist hingegen die Verwendung intern angelegter Variablen sowie das Benutzen von Informationen des den Zustandswechsel auslösenden Events – etwa das Attribut *ID* eines RFID-Events.

Für die vorliegende Arbeit sind Regelmechanismen notwendig, welche die sechs zuvor identifizierten Kontextparameter miteinbeziehen können – etwa um Benutzern ab einem bestimmten Alter eine kniffligere Fragestellung zu präsentieren als allen anderen. Einerseits werden diese Regeln direkt in die Statechart-Logik als Transition-Guards integriert – andererseits auch bei der Einbettung von kontextsensitiven Inhalten verwendet. Die folgende Auflistung zeigt Regelbeispiele, welche das System in Zukunft unterstützen soll:

- `Context.Gender == FEMALE || Context.Age < 14`
- `Context.Daytime == NIGHT && Context.Weather != RAINY`
- `Context.Temperature >= 20`
- `Context.Interests == NATURE`

Dabei ersichtlich ist, dass die Regeln jeweils aus einem Kontextparameter, einem Vergleichsoperator sowie einem statischen Referenzwert zusammengesetzt sind. Die Definition der gültigen Ausdrücke bestehend aus diesen drei Komponenten übernimmt dabei die Grammatik der Bedingungen, welche um genau diese Aspekte erweitert werden muss. Basis dafür soll Tabelle 4.3 sein, welche die zu den jeweiligen Kontextparametern passenden Operatoren und Datenbereiche festsetzt.

Bezeichnung	**Operatoren**	**Datenbereich**
`Context.Age`	`==, !=, >, <, >=, <=`	Integer > 0
`Context.Interests`	`==, !=`	`{NATURE, TECHNICS, SPORTS, HISTORY}`
`Context.Gender`	`==, !=`	`{FEMALE, MALE}`
`Context.Daytime`	`==, !=`	`{DAY, NIGHT}`
`Context.Temperature`	`==, !=, >, <, >=, <=`	Integer
`Context.Weather`	`==, !=`	`{SUNNY, CLOUDY, RAINY}`
Konjunktion	`&&`	Weitere Bedingung
Disjunktion	`||`	Weitere Bedingung

Tabelle 4.3: Definition gültiger Operatoren und Datenbereiche.

Wichtig für die zu erstellende JavaCC-Grammatik ist, dass neben den Operatoren, welche sich auf Kontextparameter beziehen, auch das Zusammenfassen mehrerer Bedingungen zu einer Regel durch Und- (`&&`) bzw. Oder-Statements (||) erlaubt wird. Durch diese Kaskadierung von Ausdrücken soll die Formulierung komplexer Regeln, welche mehrere Kontextparameter miteinbeziehen, ermöglicht werden.

4.2.5 Erweiterung der XML-Schnittstelle

Wie in Abschnitt 4.1.2 beschrieben, werden die im Admin-Tool definierten Spielszenarien als XML-Dokumente an das mobile Endgerät übertragen. Das für diesen Zweck definierte XML-Schema-Dokument verfügt bisweilen über keine Möglichkeit, über das Admin-Tool erstellte Kontext-Bedingungen zu spezifizieren. Da es bei der Erstellung von XML-Schema-Definitionen eine Vielzahl möglicher bzw. geeigneter Ansätze gibt, sei an dieser Stelle bereits auf Abschnitt 4.4 verwiesen. Darin werden u. a. das um die benötigten Aspekte erweiterte XML-Schema-Dokument sowie die Implementierung der zur XML-Generierung bzw. -Verarbeitung wichtigen Module und deren notwendige Erweiterungen näher behandelt.

4.3 Systemarchitektur

4.3.1 Gesamtsystem

Das Lomotain-System setzt sich – wie bereits in Abschnitt 4.1.2 erwähnt – aus mehreren Komponenten zusammen. Abbildung 4.9 zeigt diese Module bzw. das Gesamtsystem auf der höchsten Abstraktionsebene.

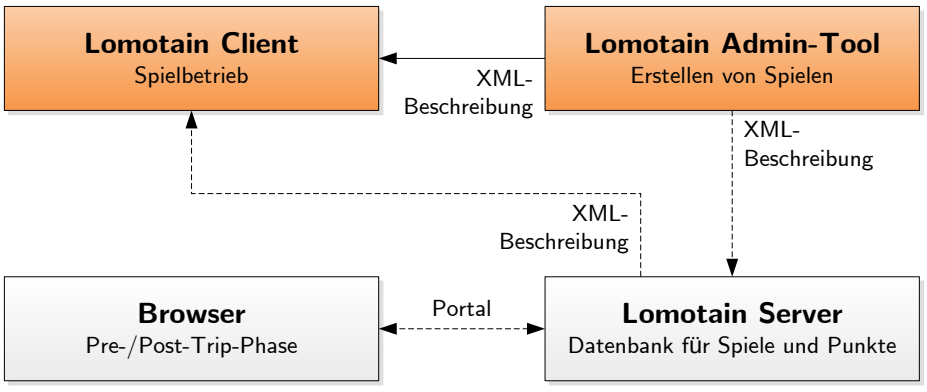


Abbildung 4.9: Überblick der Lomotain-Systemarchitektur.

Die beiden hervorgehobenen Komponenten wurden im Rahmen dieser Arbeit um kontextsensitive Konzepte erweitert. Hierzu gehört das Lomotain Admin-Tool, welches zur Erstellung von Spielszenarien konzeptioniert

ist, und der Lomotain Client. Die auf mobilen Endgeräten ausgeführte Anwendung enthält u. a. die Verarbeitung der vom Admin-Tool stammenden Spielbeschreibung sowie die Ausführung der darin enthaltenen, anhand von Statecharts definierten, Anwendungslogik. In den folgenden Abschnitten werden sowohl Client-Anwendung als auch Admin-Tool genauer vorgestellt.

4.3.2 Admin-Tool

Die zentrale Komponente zur Erstellung von Spielszenarien stellt das Lomotain Admin-Tool dar. Diese Anwendung ermöglicht es Entwicklern, die gewünschte – auf Statecharts basierende – Anwendungslogik zu definieren. Außerdem können damit alle für das jeweilige Spielszenario relevanten Inhalte eingepflegt werden. All diese Informationen werden nach Fertigstellung eines Lomotain-Abenteuers vom Admin-Tool in eine auf XML basierende Spielbeschreibung verpackt und an die Client-Anwendung übertragen.

Das ursprüngliche Lomotain Admin-Tool basiert teilweise auf Microsoft Visio und wurde in .NET/C# entwickelt. Aufgrund der Sprachinkonsistenz zur mobilen Anwendung, fehlender Flexibilität, unzureichender Wartbarkeit sowie einiger Defizite dieser ersten Version wurde die Anwendung in einem der Vorprojekte zu dieser Diplomarbeit vollständig neu entwickelt. Das neue, in Java entwickelte Admin-Tool baut auf die Eclipse RCP sowie das Modeling Framework auf und ist in Abbildung 4.10 dargestellt.

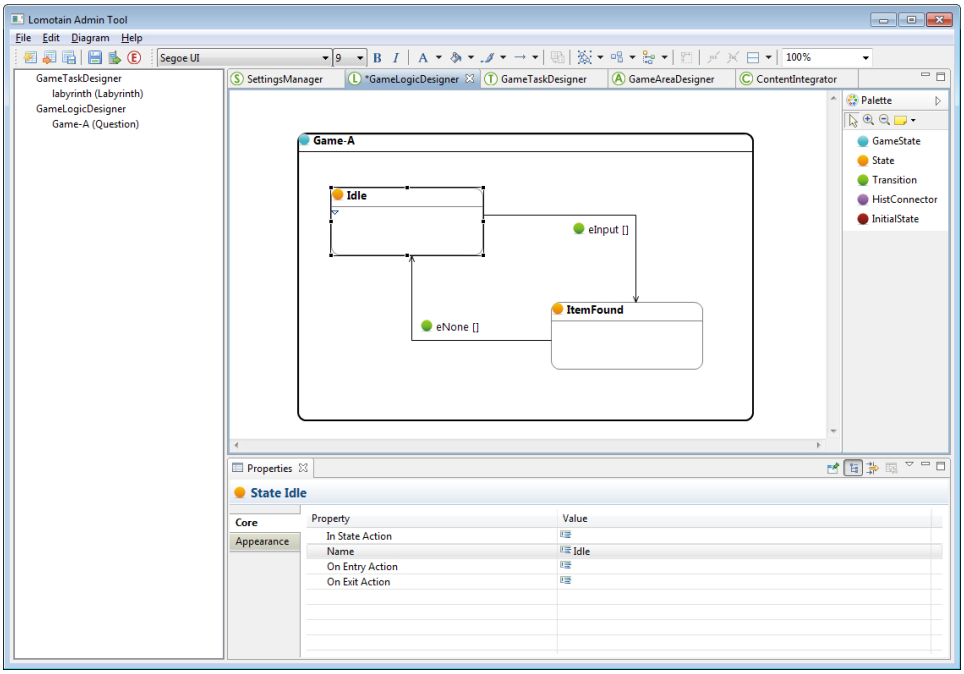


Abbildung 4.10: Screenshot des Lomotain Admin-Tools.

Den größten Bereich der Anwendung nehmen die für die Erstellung eines Spielszenarios essenziellen Module ein, welche in Form von Tabs in die Applikation integriert sind und dem Benutzer bspw. per Statechart-Editor das Definieren der gewünschten Logik erlauben. Weitere wichtige Bestandteile der Benutzeroberfläche sind der Eigenschaften-Editor (im unteren Bereich der Anwendung) sowie eine auf der linken Seite platzierte Liste, welche alle bereits erstellten Aufgaben darstellt.

Strukturell besteht das Admin-Tool aus einer Hauptanwendung sowie einigen – teils als Eclipse Plug-Ins realisierten – Subkomponenten. Abbildung 4.11 zeigt die wichtigsten Anwendungsmodule sowie die verwendeten Basistechnologien.

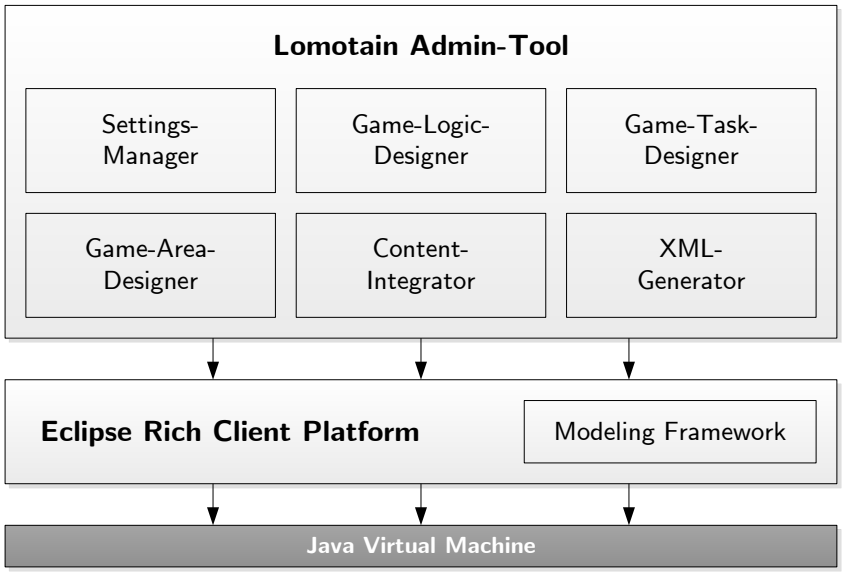


Abbildung 4.11: Struktur des Lomotain Admin-Tools.

Einige Teilkomponenten des Systems basieren neben der Rich Client Platform auch auf dem Modeling Framework. Die mit Hilfe dieser beiden Frameworks umgesetzten Module sind außerdem eigenständige Eclipse Plug-Ins und können somit bei Bedarf relativ einfach wiederverwendet werden. Alle im Blockschaltbild dargestellten Teilsysteme der modular aufgebauten Applikation übernehmen die für die Erstellung eines Spielszenarios notwendigen Aufgaben und werden im Folgenden kurz erläutert:

- **Settings-Manager:** Erlaubt das Spezifizieren diverser Grundeinstellungen (z. B. Spielname & -beschreibung).
- **Game-Logic-Designer:** Bietet dem Benutzer die Möglichkeit, die gewünschte Anwendungslogik per Statechart-Editor zu definieren. Dazu

gehört auch das Angeben diverser Aktionen, welche etwa beim Betreten eines States oder bei Zustandsübergängen ausgeführt werden sowie das Spezifizieren von Bedingungen für diese Transitions.

- **Game-Task-Designer:** Unter Verwendung vorgefertigter Templates bzw. einer im Game-Logic-Designer erstellten Logikdefinition können mit Hilfe dieses Moduls die für das Spielszenario gewünschten Teilaufgaben erstellt werden.

- **Game-Area-Designer:** Jeder Teilaufgabe wird mittels dieser Komponente und einer darin visualisierten Landkarte ein Spielareal zugeordnet. Damit ist es der Client-Anwendung später möglich, das jeweilige Rätsel im dafür vorgesehenen Bereich automatisch zu starten.

- **Content-Integrator:** Wird dazu verwendet, um zu den Aufgaben passende Inhalte in die Definition des Spielszenarios zu integrieren. Dazu gehören etwa Bilder oder Tondateien, welche beispielsweise die Aufgabenstellung enthalten.

- **XML-Generator:** Alle über die zuvor beschriebenen Module gesammelten Informationen werden von diesem Modul in eine für das mobile Endgerät lesbare Spielbeschreibung transformiert. Zum Einsatz kommt dabei eine für diesen Zweck erstellte XML-Schema-Definition, welche die Schnittstelle spezifiziert.

Weitere, in Abbildung 4.11 nicht dargestellte Bestandteile der Anwendung sind für die Verwaltung der notwendigen RFID-Tags (z. B. Spezifikation der Tag-IDs) sowie die Definition benötigter Variablen verantwortlich. Des Weiteren erlaubt es die Anwendung, im Admin-Tool definierte Spielszenarien als Projekt im Dateisystem abzulegen und zu einem späteren Zeitpunkt darauf zurückzugreifen.

Um die für den auf Statecharts basierenden Ansatz zur kontextsensitiven Anwendungsadaption benötigten Konzepte bereitstellen zu können, muss die bestehende Anwendung an verschiedenen Punkten erweitert werden. Eines der davon betroffenen Module ist der Game-Logic-Designer bzw. der darin enthaltene Statechart-Editor, welcher zukünftig auch das Definieren von auf verschiedene Kontextparameter zurückgreifende Bedingungen erlauben soll. Außerdem muss – zur Unterstützung von Ansatz 5 aus Abschnitt 4.2.3 – auch die Content-Integrator-Komponente um Funktionalitäten ergänzt werden, um kontextsensitive Bedingungen mit den definierten Inhalten verknüpfen zu können. Der für die Erstellung der Spielbeschreibung verantwortliche XML-Generator muss ebenso weiterentwickelt werden, um neben der herkömmlichen Spielbeschreibung in Zukunft auch alle für den kontextsensitiven Ansatz notwendigen Informationen ins erstellte XML-Dokument verpacken zu können.

4.3.3 Client-Anwendung

Den mobilen Gegenpart zum Admin-Tool verkörpert die Lomotain Client-Anwendung. Diese erhält die mit Hilfe des Admin-Tools erstellte Spielbeschreibung (z. B. per Bluetooth), parst diese und erzeugt eine interne Datenstruktur mit allen benötigten Informationen, um das im XML-Dokument definierte Spielszenario ausführen zu können. Die Spielausführung stellt die Hauptaufgabe der Applikation dar und begleitet den Benutzer anhand der jeweiligen Aufgaben durch das gesamte Lomotain-Abenteuer. Ein ebenfalls wichtiger Aspekt der Client-Anwendung ist die Interaktion mit dem Benutzer bzw. mit der Umwelt, wobei das System bereits verschiedenen Technologien unterstützt.

Die aus dem ursprünglichen Lomotain-Projekt stammende Version der Client-Anwendung wurde in Java ME (Java Mobile Edition) entwickelt und beinhaltet bereits alle für die Ausführung eines Spielszenarios relevanten Komponenten. Aufgrund unzureichender Modularität sowie neuer Anforderungen wurde diese erste Version im Rahmen eines Vorprojekts dieser Diplomarbeit hinsichtlich Systemarchitektur und Interaktionsmechanismen weiterentwickelt. Viele Basismodule der Anwendung blieben dabei jedoch bestehen – u. a. diverse Screens, welche zur Ausführung der Spielszenarien benötigt werden. Einige davon zeigt Abbildung 4.12.

Abbildung 4.12: Screenshots der Lomotain Client-Anwendung.

Die abgebildeten Screens zeigen den üblichen Ablauf einer Spielausführung. Beginnend beim Hauptmenü wählt der Benutzer u. a. das gewünschte Abenteuer – vom dafür zuständigen Game-Screen wird anschließend die jeweilige Aufgabe visualisiert. Wurde das Rätsel erfolgreich gelöst, wird von einem Success-Screen die dazu passende Erfolgsmeldung illustriert.

Ähnlich wie das Admin-Tool ist auch die Client-Anwendung in diverse Subkomponenten gegliedert, welche jeweils wichtige – zur Spielausführung notwendige – Aufgaben übernehmen. Abbildung 4.13 zeigt die interne Struktur der Applikation mitsamt den bedeutendsten Modulen.

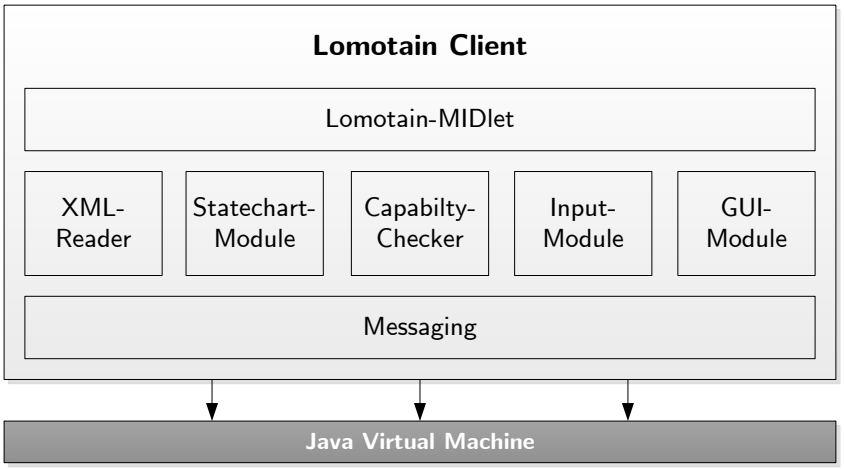


Abbildung 4.13: Struktur der Lomotain Client-Anwendung.

Kern der Anwendung ist eine sogenannte MIDlet-Klasse, welche als Einsprungspunkt dient und bspw. das Visualisieren eines Start-Screens bzw. des Hauptmenüs veranlasst. Außerdem werden von dieser Komponente alle weiteren Module instanziiert sowie verwaltet. Diese essenziellen Systembestandteile werden im Folgenden kurz erläutert.

- **XML-Reader:** Liest die vom Admin-Tool stammende Spielbeschreibung ein, prüft diese auf Korrektheit und erstellt u. a. aus der Logik-Beschreibung eine interne Datenstruktur zur weiteren Verarbeitung.
- **Statechart-Module:** Die vom XML-Reader angelegten Datenstrukturen nutzt u. a. dieses Modul, um das gesamte Spielszenario auszuführen, Inhalte anzuzeigen sowie Eingaben (z. B. mittels RFID) entgegenzunehmen. Teilbereich dieser Komponente ist u. a. die JavaCC-Grammatik, welche zur Verarbeitung von Transition-Guards (Bedingungen der Zustandsübergänge) und später benötigter, kontextsensitiver Bedingungen der Content-Objekte dient.
- **Capability-Checker:** Überprüft anhand der gegebenen Hardware-Ressourcen die Anwendbarkeit der diversen Interaktionsmechanismen und GUI-Konzepte. Nur jene Technologien, welche das jeweilige Mobiltelefon unterstützt, werden dem Benutzer angeboten.
- **Input-Module:** Beinhaltet die derzeit zur Interaktion unterstützten Mechanismen (RFID, 2D-Code, Tastatureingabe).
- **GUI-Module:** Inkludiert eine mittels simpler Java ME-Technologien erstellte Benutzeroberfläche.

- **Messaging:** Der als zentraler Kommunikationspunkt genutzte Mechanismus ermöglicht den Informationsaustausch zwischen den einzelnen Komponenten.

Die bestehende Client-Anwendung besitzt im aktuellen Zustand keine Fähigkeit, kontextsensitive Adaptionen durchzuführen. Hierzu müssen – ähnlich wie beim Admin-Tool – an mehreren Stellen Erweiterungen durchgeführt werden. Dazu gehört einerseits der XML-Reader, welcher die hinzukommenden, kontextsensitiven Informationen einlesen und in die interne Datenstruktur integrieren muss. Andererseits ist es notwendig, das Statechart-Modul in Richtung Kontextsensitivität weiterzuentwickeln. Wichtigster Aspekt dabei ist die Unterstützung von Transition-Guards, welche einen oder mehrere Kontextparameter miteinbeziehen können. Die Verarbeitung dieser neuen, kontextsensitiven Bedingungen erfordert wiederum Erweiterungen der Client-Anwendung. Hierzu zählt sowohl ein Modul zum Erfassen der aktuellen Kontextinformationen (z. B. Benutzerprofil) als auch eine Management-Komponente, welche die erhaltenen Daten an die jeweiligen Anwendungsbereiche weiterleitet.

4.4 Implementierung

4.4.1 Admin-Tool

Das Lomotain Admin-Tool wird zur Definition von Spielszenarien genutzt und wurde in Java unter Verwendung der Eclipse Rich Client Platform sowie des Eclipse Modeling Frameworks implementiert. Abbildung 4.14 illustriert die grundlegende Packagestruktur dieser Desktop-Applikation.

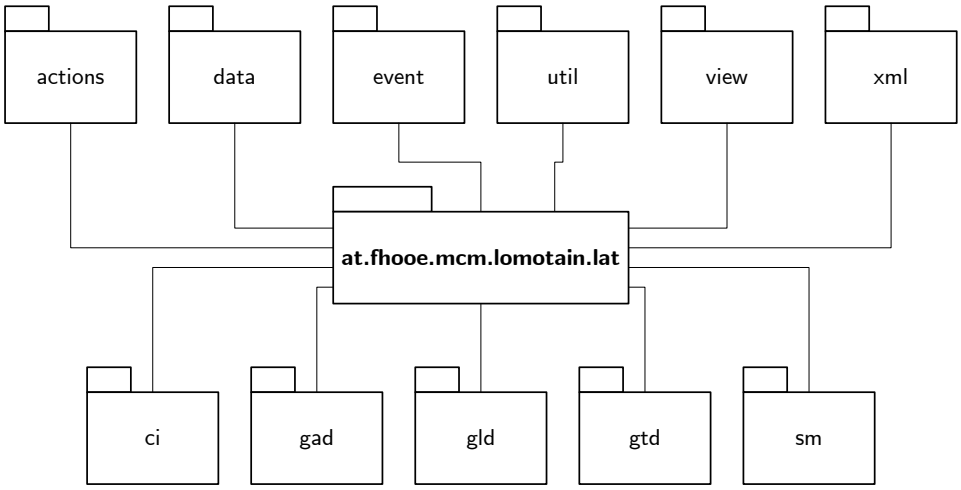


Abbildung 4.14: Packagestruktur des Admin-Tools.

Die im oberen Bereich der Abbildung dargestellten Packages sind fester Bestandteil der zentralen RCP-Anwendung (`at.fhooe.mcm.lomotain.lat`) – alle weiteren Komponenten wurden als Eclipse Plug-Ins umgesetzt und erweitern somit als selbständige Module die Basisanwendung um die benötigten Funktionalitäten. Die folgenden Abschnitte beschreiben die wichtigsten Bereiche der einzelnen Packages bzw. Plug-Ins sowie die für die kontextsensitiven Aspekte umgesetzten Erweiterungen.

RCP-Basisanwendung

Die zentrale Komponente des Admin-Tools stellt eine auf die Eclipse Rich Client Platform basierende Anwendung dar, welche ausgehend von einer existierenden Beispielapplikation entwickelt wurde und aus mehreren Komponenten zusammengesetzt ist. Startpunkt der Anwendung ist die Klasse `Application`, welche die sogenannte Eclipse-Workbench mit Hilfe eines `WorkbenchAdvisors` startet. Dieser nutzt einen `WorchbenchWindowAdvisor` sowie die Klasse `Perspective`, um die zur Anwendung gehörenden Fenster zu konfigurieren bzw. das Layout der gesamten Anwendung (z. B. Position der Views und Editors) festzulegen. Auch können innerhalb der `Perspective`-Klasse bereits erstellte Views eingebunden werden – die Aufgabenliste wurde dem linken Anwendungsbereich des Admin-Tools auf diese Art hinzugefügt.

Die innerhalb der RCP-Anwendung benötigten Aktionen (Einträge in Menü- und Symbolleiste) werden in der Klasse `ActionBarAdvisor` definiert. Dabei wurden im Rahmen dieses Projekts etwa Aktionen zur Verwaltung von Lomotain-Projekten (Neu, Öffnen, Speichern) sowie zur Erstellung der XML-Spielbeschreibung umgesetzt, welche jeweils eine generische `Action`-Klasse erweitern und in einer `run()`-Methode die gewünschte Funktionalität implementieren.

Zur internen Kommunikation zwischen verschiedenen Komponenten bzw. Plug-Ins wurde die RCP-Applikation um einen Event-Mechanismus erweitert. Die dafür zuständigen Klassen visualisiert Abbildung 4.15.

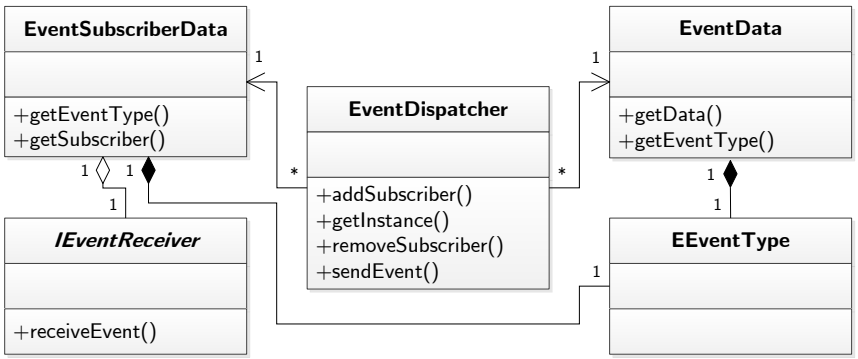


Abbildung 4.15: Klassenstruktur des internen Kommunikationskonzepts.

Dabei können sich interessierende Module mit Hilfe einer dafür vorgesehenen Datenstruktur (`EventSubscriberData`) am zentralen `EventDispatcher` anmelden. Alle über diese Klasse veröffentlichten Events (Objekte vom Typ `EventData`) werden sodann an die jeweiligen, für diesen Event-Typ registrierten, Komponenten weitergeleitet. Jene Module, welche Informationen über diesen Mechanismus erhalten möchten, müssen das für diesen Zweck entwickelte Interface `IEventReceiver` implementieren.

Die für die Vorhaltung der vom Benutzer angegebenen Informationen (Inhalte, Logikdefinition etc.) genutzte interne Datenstruktur ist ebenso Bestandteil der RCP-Basisanwendung und wird neben dieser auch von allen im weiteren Verlauf vorgestellten Plug-Ins verwendet. Die Datenstruktur besteht aus mehreren Klassen, welche in Abbildung 4.16 dargestellt sind.

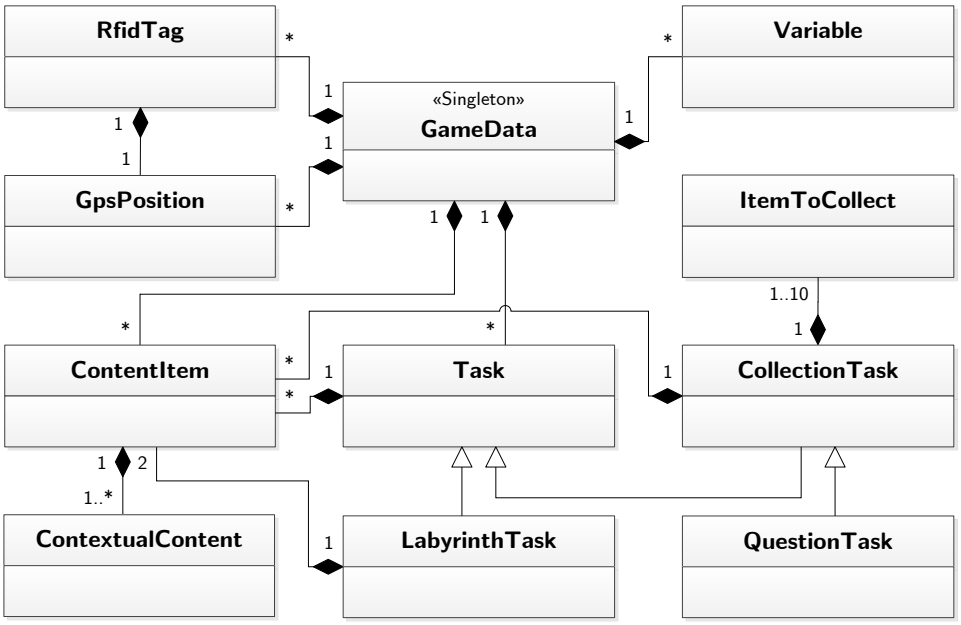


Abbildung 4.16: Datenstruktur zur Beschreibung von Spielszenarien.

Mit Hilfe der abgebildeten Klassenstruktur ist es möglich, gesamte Spielszenarien inklusive Logik und Inhaltsinformationen zu beschreiben. Zentrale Komponente ist die Klasse `GameData`, welche alle relevanten Daten beinhaltet und nach dem Singleton-Entwurfsmuster [58] implementiert wurde. Ebenso ersichtlich ist die Trennung der Aufgaben in drei Typen (Labyrinth, Sammeln, Fragestellung), wobei `QuestionTask` große Ähnlichkeit mit `CollectionTask` aufweist und daher von dieser Klasse abgeleitet ist.

Wichtigste Erweiterung im Hinblick auf Kontextsensitivität ist die Einführung der Klasse `ContextualContent`. Jedes `ContentItem`-Objekt besitzt mindestens eine Instanz davon und erweitert dieses Element um zumindest eine – mit kontextsensitiven Bedingungen versehene – Inhaltsdefinition (z. B.

Pfad zu einem Bild). Das Konzept – verwendet in mehreren Bereichen der Datenstruktur – löst die früher direkt im `ContentItem` hinterlegten Inhaltsdefinitionen davon und transferiert diese in die neue Klasse. Zur Laufzeit ist es somit dem `ContentItem`-Objekt möglich, jenes `ContextualContent`-Element auszuwählen und zu verwenden, welches optimal zur aktuellen Umgebungssituation passt.

XML-Generator

Ein in der RCP-Basisanwendung enthaltenes, jedoch relativ eigenständiges Modul ist der im Package `xml` vorhandene XML-Generator. Die diese Komponente implementierende Klasse `GameSpecificationGenerator` nutzt vor allem die soeben vorgestellte Datenstruktur und die XML-Library JAXP, um aus den vorhandenen Informationen eine für die Client-Anwendung gültige Spielbeschreibung zu erzeugen. Die Generierung der XML-Datei erfolgt in verschiedenen Schritten, welche jeweils für die Einbindung von Logik-, Inhalts-, Areal-, RFID- sowie allgemeinen Informationen verantwortlich sind.

Für die Definition der Anwendungslogik von Spielszenarien gibt es zwei Möglichkeiten – entweder die Verwendung vorgefertigter Templates, oder die Spezifikation der Logik per Game-Logic-Designer. Dieser – mit Hilfe des Eclipse Modeling Projects automatisch generierte – Editor legt die angegebenen Informationen bzw. das darin definierte Statechart intern wiederum als XML-Datei ab. Dieses wird zur Generierung der eigentlichen Spielbeschreibung benötigt und muss für diesen Zweck zuerst eingelesen werden. Abbildung 4.17 stellt die im Rahmen der XML-Generierung notwendigen Aktionen dar.

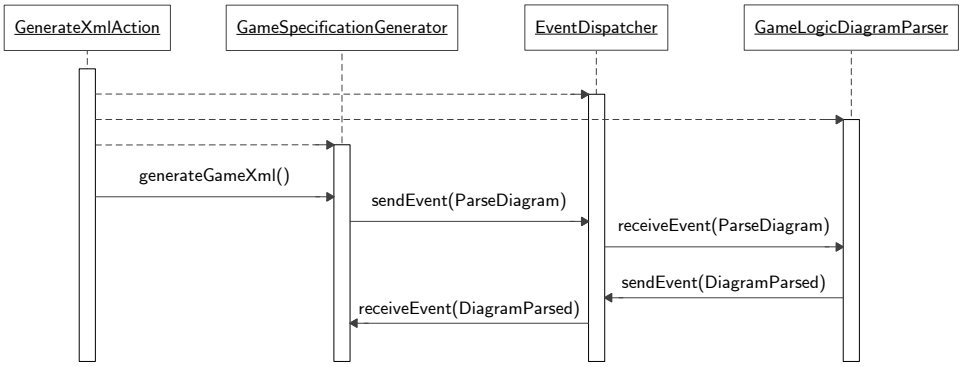


Abbildung 4.17: Erstellung der XML-Spielbeschreibung.

Ausgehend von einer über die Rich Client Platform definierten Aktion werden dabei alle Instanzen der zur XML-Erzeugung benötigten Klassen angelegt. Über den internen Event-Mechanismus wird sodann ein Parser für die automatisch vom Game-Logic-Designer erstellte XML-Datei benachrichtigt.

Nach erfolgreichem Einlesen des XML-Dokuments notifiziert dieser Parser wiederum den `GameSpecificationGenerator`, um die Weiterführung der eigentlichen XML-Generierung anzustoßen.

Der XML-Generator transferiert die benötigten und somit alle in der internen Datenstruktur vorhandenen Informationen in ein XML-Dokument. Da die verwendete Datenstruktur um den bereits beschriebenen kontextsensitiven Aspekt (Klasse `ContextualContent`) erweitert wurde, musste auch das für die XML-Erzeugung zuständige Modul dahingehend angepasst werden. Diese Weiterentwicklung des XML-Generators betrifft in erster Linie jenen Bereich, welcher die Inhaltsobjekte enthält. Darin werden nun – ähnlich der Struktur im Klassendiagramm – jedem `ContentItem`-Objekt Kindelemente mit den Informationen der zugehörigen `ContextualContent`-Elemente angefügt und somit an die Client-Anwendung übertragen.

Content-Integrator & Settings-Manager

Die zur Erfassung allgemeiner bzw. inhaltsspezifischer Informationen verwendeten, ähnlich aufgebauten Komponenten Settings-Manager und Content-Integrator wurden als eigenständige Eclipse Plug-Ins implementiert. Dies hat den Vorteil, dass die Komponenten modular getrennt und somit leicht austauschbar sind, und trotzdem relativ einfach in die RCP-Basisanwendung integriert werden können. Abbildung 4.18 zeigt die zum Content-Integrator gehörenden Klassen sowie die von diesem Modul benutzten Komponenten.

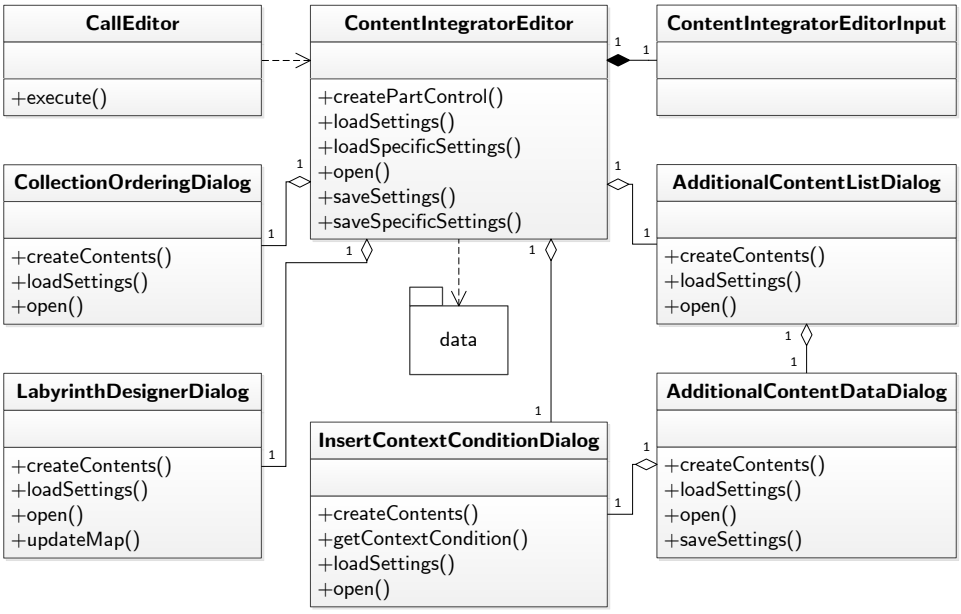


Abbildung 4.18: Klassendiagramm des Content-Integrators.

Ausgehend von einer `CallEditor`-Klasse wird der eigentliche Editor – definiert in `ContentIntegratorEditor` – gestartet. Dieser benutzt einerseits zum Abspeichern der diversen Informationen die Objekte des Packages `data`, andererseits eine Reihe weiterer Dialogfenster – etwa zum Definieren der Labyrinth-Struktur oder zum Angeben global benötigter Inhaltsobjekte.

Die im Content-Integrator notwendigen Erweiterungen hinsichtlich Kontextsensitivität beziehen sich vor allem auf den zentralen Editor und die Klasse `AdditionalContentDataDialog`. Die bedeutendste Veränderung stellt die Erweiterung hinsichtlich der Definition von Kontextsituationen für Inhaltsobjekte (Bilder, Musik etc.) dar, welche mit Hilfe eines neuen Dialogfensters (`InsertContextConditionDialog`) umgesetzt wurde. Der Benutzer kann dadurch für ein Inhaltsobjekt mehrere Kontextsituationen mitsamt Inhaltsdefinitionen einpflegen, welche später zur Laufzeit berücksichtigt werden.

Game-Logic-Designer

Der mit Hilfe des EMF (Eclipse Modeling Framework) sowie des GMF (Graphical Modeling Framework) erstellte Game-Logic-Designer verkörpert ebenso ein eigenständiges Eclipse Plug-In und beruht auf einem sogenannten Domain-Model, welches die Basiskomponenten des späteren Statechart-Editors definiert und in Abbildung 4.19 dargestellt ist.

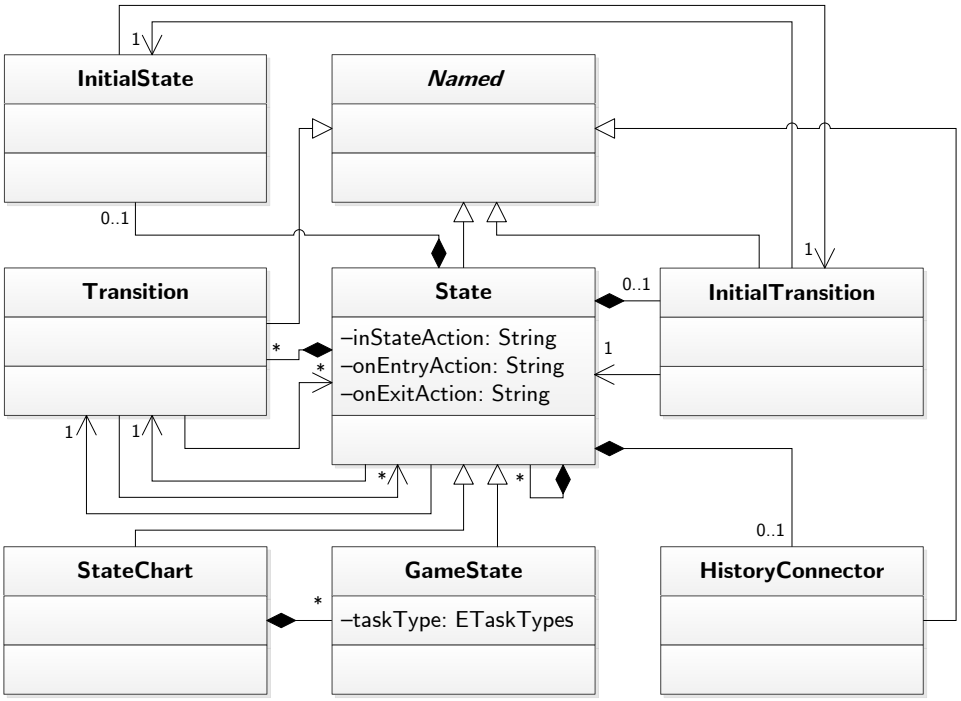


Abbildung 4.19: Domain-Model des Game-Logic-Designers.

Ähnlich zu UML-Klassendiagrammen enthält das Domain-Model die für die jeweilige Anwendungsdomäne wichtigen Objekte – in diesem Fall die einzelnen Bausteine des gewünschten Statechart-Editors. Dazu gehören allgemeine States und Transitions sowie spezielle Formen davon. Ebenfalls wichtig sind die Module `StateChart` sowie `GameState`, welche das gesamte zukünftige Statechart-Diagramm bzw. States, welche die Anwendungslogik einer gesamten Aufgabe repräsentieren, darstellen.

Großer Vorteil der verwendeten Frameworks EMF und GMF ist die automatische Generierung vollständiger Editoren bzw. Eclipse Plug-Ins auf Basis des bereits definierten Domain-Models. Die für diesen Zweck notwendigen Arbeitsschritte wurden bereits in Abschnitt 3.2.2 kurz vorgestellt und werden im Folgenden genauer erläutert:

1. Erstellung eines **Generator-Models** zur Generierung eines textbasierten Editors, welcher neben dem Domain-Model die Grundlage des gewünschten grafischen Statechart-Editors bildet. Das Generator-Model kann dabei mit Hilfe eines Wizards als neues Dokument angelegt und mit dem Domain-Model verknüpft werden.

2. Die **Generierung des textbasierten Editors** erfolgt über den Wurzelknoten des Generator-Models. Dabei werden von Eclipse diverse Source-Code-Verzeichnisse sowie Projekte angelegt – darunter eines mit der Endung `editor`. Dieses kann bereits als Eclipse-Anwendung gestartet werden und enthält u. a. die Möglichkeit, eine Instanz des Domain-Models per textbasiertem Editor zu erstellen.

3. Definition der grafischen Darstellung, welche die Visualisierungsaspekte der im grafischen Editor gewünschten Objekte (z. B. abgerundetes Rechteck zur Darstellung von States) beinhaltet. Dieses **Graphical-Definition-Model** wird unabhängig von den bisherigen Dokumenten erstellt und soll nach Fertigstellung für jedes gewünschte Objekt des Domain-Models eine Visualisierungsdefinition enthalten.

4. Erstellung des **Tooling-Definition-Models** – dieses wird ebenfalls als eigenständiges Dokument erstellt und enthält alle später in einer Tooling-Palette bzw. einem Werkzeugkasten verfügbaren Komponenten, welche der Benutzer im grafischen Editor verwenden kann, um die Anwendungslogik zu definieren (z. B. State, Transition).

5. Erstellung des **Mapping-Models**, welches die beiden soeben erstellten Dokumente und das ursprüngliche Domain-Model zusammenführt. Dabei werden mit Hilfe eines Wizards die Objekte der grafischen Definition sowie der Tooling-Palette mit den Elementen des Domain-Models verknüpft. Somit ist es dem Framework bei der Generierung des grafischen Editors möglich, die notwendigen Assoziationen herzustellen.

6. Erstellung eines **Generator-Models** zur Generierung des grafischen Editors. Ähnlich wie zuvor wird auch für den grafischen Editor ein Generator-Model benötigt, welches wiederum per Wizard mit den notwendigen Informationen versehen wird und die automatische Code-Generierung ermöglicht.

7. Die **Generierung des grafischen Editors** erfolgt ähnlich wie der erste Code-Generierungs-Schritt über das Kontextmenü des soeben definierten Generator-Models. Dabei wird wiederum ein neues Eclipse-Projekt (mit der Endung `diagram`) erstellt, welches einige der zuvor generierten Projekte referenziert und den grafischen Statechart-Editor enthält. Genau wie der textbasierte kann auch der grafische Editor sofort nach dem Generierungsprozess gestartet und verwendet werden.

Im Rahmen der automatischen Code-Generierung ist es meist nicht möglich, alle gewünschten Funktionen und Anforderungen zu berücksichtigen. Deshalb war es auch während der Entwicklung des Admin-Tools notwendig, einige Anpassungen im Nachhinein manuell einzupflegen sowie Erweiterungen für die erstellte Applikation zu implementieren. Dazu gehört etwa ein Dialogfenster, welches dem Benutzer die komfortable Integration von möglichen Methodenaufrufen mitsamt Parameterübergabe innerhalb von On-Entry-Actions ermöglicht. Außerdem gab es im Hinblick auf kontextsensitive Aspekte Grund zur Weiterentwicklung. Das wichtigste neue Konzept stellt dabei die Integration von kontextsensitiven Ausdrücken innerhalb der Guard-Definition dar, welche durch ein bereits im Content-Integrator verwendetes Dialogfenster (`InsertContextConditionDialog`) ermöglicht wurde und in Abbildung 4.20 dargestellt ist.

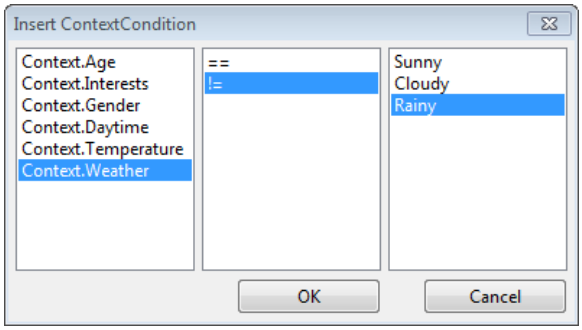


Abbildung 4.20: Dialog zur Integration kontextsensitiver Bedingungen.

Game-Task-Designer

Ähnlich wie der Game-Logic-Designer wurde auch diese Komponente mittels den Frameworks EMF und GMF umgesetzt und basiert ebenso auf einem

Domain-Model, welches die verschiedenen Aufgabentypen (Labyrinth, Sammeln, Fragestellung) sowie ein Transition-Objekt zur Festlegung der Reihenfolge definiert und in Abbildung 4.21 visualisiert ist.

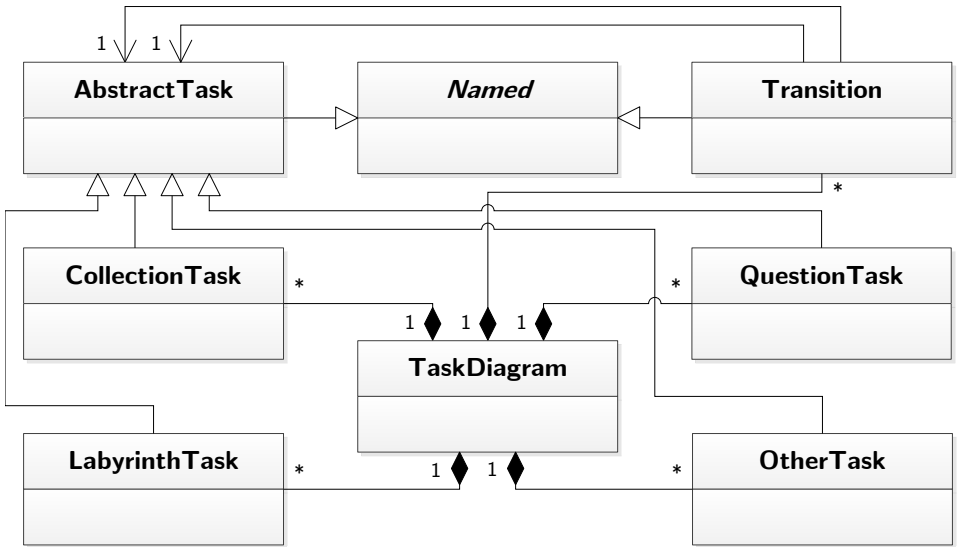


Abbildung 4.21: Domain-Model des Game-Task-Designers.

Die Umsetzung des Game-Task-Designers erfolgte äquivalent zur Entwicklung des Game-Logic-Designers und ermöglicht dem Benutzer die konzeptionelle Definition von Aufgaben eines gesamten Spielszenarios. Die einzelnen Abenteuer können bei Bedarf in einer bestimmten Reihenfolge spezifiziert und mit vorgegebenen Logik-Templates oder einer selbst erstellten Logik-Definition aus dem Game-Logic-Designer versehen werden.

Game-Area-Designer

Das zur Festlegung der Aufgaben- bzw. Spielareale notwendige Modul Game-Area-Designer stellt ein eigenständiges Eclipse Plug-In dar und nutzt die `JXMapViewer`-Komponente der frei verfügbaren Library swingx-ws[5] zur Visualisierung von OpenStreetMap[6]-Kartenmaterial. Dem Benutzer wird es dadurch ermöglicht, für jede Aufgabe sowie das gesamte Spielszenario einen Bereich – definiert durch Koordinaten – abzustecken. Diese Areal-Informationen werden in einem ersten Schritt in der internen Datenstruktur abgelegt sowie später zur Erstellung der XML-Spielbeschreibung herangezogen. Am mobilen Endgerät ist es der Client-Anwendung zur Laufzeit somit möglich, durch Eruierung der GPS-Koordinaten die jeweils für den aktuellen Bereich definierte Aufgabe vollautomatisch zu starten.

[5] https://swingx-ws.dev.java.net
[6] http://www.openstreetmap.org

4.4.2 Client-Anwendung

Die Lomotain Client-Anwendung wurde in Java ME implementiert und ermöglicht es bspw. Besuchern eines Themenparks, die dort verfügbaren Spielszenarien auszuführen und damit gesteckte Ziele (z. B. spielerisches Kennenlernen der Natur) zu erreichen. Die Anwendung verwendet CLDC (Connected Limited Device Configuration) 1.1 sowie MIDP (Mobile Information Device Profile) 2.0, wurde vorrangig mit den NFC-kompatiblen Mobiltelefonen Nokia 6131 bzw. Nokia 6212 entwickelt und gliedert sich in diverse Packages, welche in Abbildung 4.22 dargestellt sind.

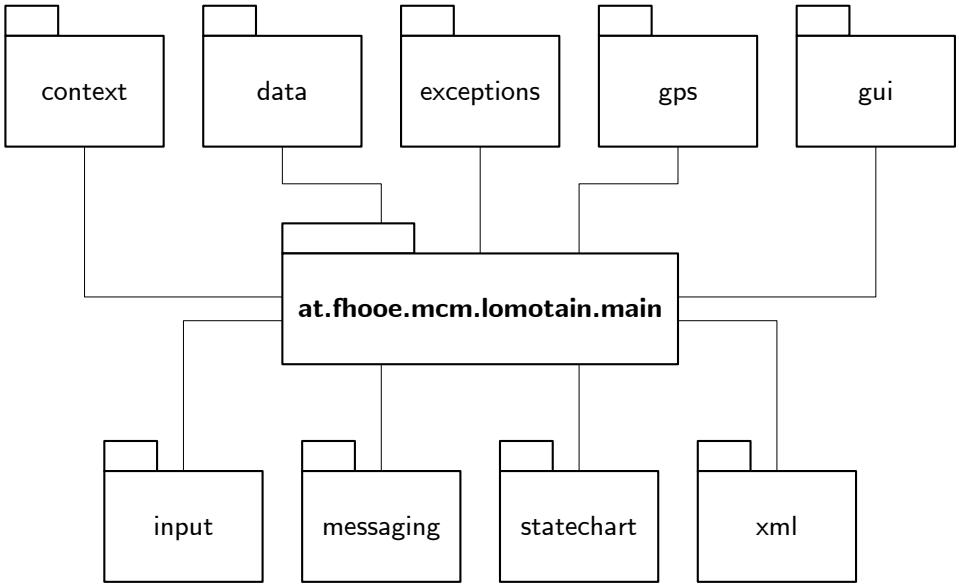


Abbildung 4.22: Packagestruktur der Lomotain Client-Anwendung.

Die in einem Vorprojekt zu dieser Diplomarbeit weiterentwickelte mobile Anwendung besitzt mit dem `main`-Package eine zentrale Komponente, welche u. a. das MIDlet enthält, welches die gesamte Anwendung startet sowie ein Hauptmenü spezifiziert. Des Weiteren besitzt das Package einen `CapabilityChecker`, welcher die Aufgabe hat, die vorhandenen Hardware-Ressourcen zu ermitteln und dem Benutzer etwa nur die am aktuell verwendeten Mobilgerät vorhandenen Interaktionsmechanismen (z. B. RFID) zur Verfügung zu stellen.

Neben der Hauptkomponente gibt es eine Reihe weiterer Packages – einige davon sind für die gewünschten kontextsensitiven Aspekte nicht zwingend erforderlich, erfüllen jedoch andere wichtige Aufgaben:

- **data**: Enthält ähnlich wie das Admin-Tool eine interne Datenstruktur, welche die vom XML-Reader eingelesenen Informationen eines gesamten Spielszenarios repräsentiert. Bei der Entwicklung wurde zum

Großteil auf die Klassenstruktur des Admin-Tools zurückgegriffen, um mögliche zukünftige Änderungen flexibler und einfacher einpflegen zu können.

- **exceptions**: Beinhaltet einige, vor allem für das Statechart-Modul benötigte Exception-Klassen.
- **gps**: Wird zwar in der aktuellen Anwendungsversion nicht verwendet, bietet jedoch die Möglichkeit, einen am Mobilgerät vorhandenen GPS-Empfänger anzusprechen und die aktuellen Koordinaten abzufragen.
- **gui**: Implementiert einerseits alle für die jeweiligen Aufgabentypen benötigten, andererseits auch diverse allgemeine Screens (z. B. Status, Map, Start etc.). Die aktuell mit standardmäßig in Java ME vorhandenen Komponenten erstellte Benutzeroberfläche kann aufgrund der modularen Struktur zukünftig z. B. durch ein mittels Flash Lite[7] oder JavaFX[8] entwickeltes GUI ersetzt werden.
- **input**: Enthält alle derzeit implementierten Interaktionsmechanismen (RFID, 2D-Code, Tastatureingabe). Aufgrund einer modularen Entwicklung sowie des zur Implementierung gewählten Factory-Entwurfsmusters [58] ist es relativ einfach möglich, die Anwendung um neue Bedienkonzepte (z. B. Gestenerkennung) zu ergänzen.
- **messaging**: Ähnlich wie im Admin-Tool wird auch hier ein Mechanismus zum Austausch von Informationen benötigt – dieser wird etwa zum Verteilen von Events (z. B. für die Benutzerinteraktion) verwendet.

Alle weiteren, noch nicht behandelten Packages (`context`, `statechart`, `xml`) wurden teilweise zur Integration kontextsensitiver Aspekte neu erstellt bzw. um die benötigten Konzepte erweitert und werden im Folgenden genauer behandelt.

Kontext-Distribution

Grundlegende Informationen, um kontextsensitive Anwendungen erstellen zu können, sind die diversen – für das vorliegende Projekt in Abschnitt 4.2.2 identifizierten – Kontextparameter. Die Erfassung dieser übernimmt dabei ein innerhalb des `main`-Packages implementiertes Formular, das es dem Anwender erlaubt, die sechs spezifizierten Parameter (Alter, Interessen, Geschlecht, Tageszeit, Temperatur, Wetter) mit den gewünschten Informationen zu füllen. Dieses Benutzerprofil ist über das allgemeine Hauptmenü erreichbar und sollte vom Benutzer vor dem Start eines Spielszenarios aktualisiert werden.

[7] http://www.adobe.com/products/flashlite
[8] http://www.javafx.com

Das zur Distribution der so erfassten Informationen neu implementierte Modul (Package `context`) besteht u. a. aus einem `ContextDispatcher`. Diese nach dem Singleton-Entwurfsmuster [58] entwickelte Klasse bietet einerseits die Möglichkeit, bei Änderung im Benutzerprofil die neuen Informationen abzulegen. Andererseits können alle interessierenden Module jederzeit auf die aktuellsten Werte zurückgreifen und somit die Anwendung kontextsensitiv anpassen.

XML-Reader

Die Gegenkomponente des im Admin-Tool vorhandenen XML-Generators stellt der XML-Reader `GameSpecificationParser` (Package `xml`) der Client-Anwendung dar. Dieser nutzt die kXML-Library und ist für das Einlesen des das Spielszenario beschreibenden XML-Dokuments verantwortlich. Die einzelnen Teilbereiche des XML-Readers verarbeiten die unterschiedlichen Bereiche des XML-Dokuments (Logik-, Inhalts-, Areal-, RFID- und allgemeiner Abschnitt). Alle so erhaltenen Informationen werden in die dafür vorgesehene Datenstruktur (Package `data`) eingespeist, welche zur Laufzeit des Spielszenarios von der Anwendung Verwendung findet.

Genau wie der XML-Generator wurde im Rahmen dieses Projekts auch der XML-Reader an die neue XML-Schnittstelle angepasst, welche nun neben den ursprünglichen Daten auch Informationen zur kontextsensitiven Adaption der Anwendung enthält. Dazu zählen etwa die mit Hilfe von Kontextparametern erstellten Bedingungen der einzelnen Inhaltsobjekte.

Statechart-Modul

Das für die kontextsensitiven Aspekte wichtigste Modul stellt das Package `statechart` dar, welches für die Ausführung der auf Statecharts basierenden Anwendungslogik zuständig ist. Neben den für die Statecharts essenziellen Klassenstrukturen (State, Transition, Action, Guard etc.) enthält dieses Package weitere Komponenten, welche dem ursprünglichen Lomotain-Projekt [22, 48] entstammen und nahezu unverändert übernommen wurden. Dazu zählen die Implementierungen diverser in der Statechart-Logik benötigter Events (z. B. Input, AreaEntered, AreaLeft) sowie ein Memory-Modul, welches in der Logik definierte Variablen (z. B. für Punktestand) verwaltet.

Zu den wichtigsten Aufgaben dieses Moduls gehört außerdem das Parsen und Verarbeiten der in den Statechart-Transitions definierten Guards sowie den spezifizierten State-Actions (z. B. On-Entry-Action). Dafür wurden mit Hilfe von JavaCC bereits im ursprünglichen Lomotain-Projekt zwei Parser erstellt (`ConditionLanguageParser` sowie `ActionLanguageParser`). Da im Rahmen dieses Projekts keine Veränderung an den vorhandenen Actions durchgeführt wurde, konnte letzterer Parser vollständig übernommen werden. Zur Verarbeitung der um die diversen Kontextparameter erweiterten

Transition-Guards wurde hingegen eine neue Grammatik definiert, welche ebenfalls zum Prüfen der kontextsensitiven Bedingungen von Inhalten verwendet wird und neben den ursprünglichen Elementen auch die Verwendung der neuen Kontextparameter ermöglicht. Einige der Token, welche die neu definierte Sprache `ContextConditionLanguage` verwendet, zeigt Tabelle 4.4.

Token	**Wertebereich**
`<EQUALS>`	`("==")`
`<EQUALSNOT>`	`("!=")`
`<WEATHER>`	`("SUNNY", "CLOUDY", "RAINY")`
`<CONTEXT_WEATHER>`	`("Context.Weather")`

Tabelle 4.4: Token der neu erstellten Guard-Grammatik (Auszug).

Die dargestellten Tokendefinitionen für den Kontextparameter *Wetter* inkludieren sowohl die aus der Konzeption (Abschnitt 4.2.2) übernommenen Operatoren als auch alle möglichen Werte dieses Kontextparameters.

Nach der Definition der gültigen Token für die Grammatik können ebenfalls im JavaCC-Dokument die gewünschten Regeln erstellt werden, welche die korrekte Syntax der späteren Bedingungen festlegen. Einige davon zeigt Abbildung 4.23.

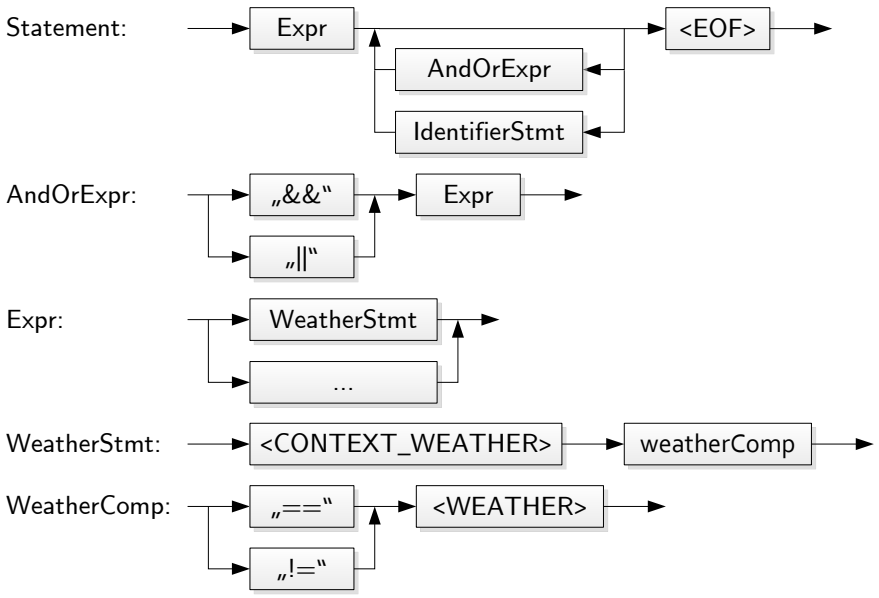


Abbildung 4.23: Syntaxdiagramm der neuen Guard-Grammatik (Auszug).

Dabei ist ersichtlich, dass die Grammatik mit einem `Statement` beginnt, welches aus einer Expression (`Expr`) sowie weiterer optionaler Konstrukte besteht. Die einzelnen, die Kontextparameter darstellenden Syntaxregeln verbergen sich hinter dem `Expr`-Objekt und definieren sich in jeweils eigenständigen Statements (z. B. `WeatherStmt`). Die in der Abbildung dargestellte Struktur hat sich als eine der möglichen Umsetzungsvarianten erwiesen. Dabei setzt sich die Grammatikregel aus dem jeweiligen Kontextparameter (z. B. `<CONTEXT_WEATHER>`), einem Operator (z. B. `==`) sowie einem Vergleichswert (z. B. `<WEATHER>`) zusammen.

Für jede der hier dargestellten Syntaxregeln wurde im JavaCC-Dokument eine Methode definiert, welche die gewünschte Struktur festlegt und dadurch die Eingaben hinsichtlich Korrektheit der Syntax prüft. Außerdem wird bereits in diesen Methoden mit der Erzeugung einer Baumstruktur der Grundstein für die spätere Verarbeitung der Ausdrücke gelegt. Eine der Grammatikregeln mitsamt Anweisungen zur Erstellung dieser internen Datenstrukturen zeigt Listing 4.1.

```
1  TreeNode weatherStmt() : {
2    TreeNode root;
3  } {
4    <CONTEXT_WEATHER> root = weatherComparison(
5      new TreeNodeContextElement(ContextParameter.Weather))
6    {
7      return root;
8    }
9  }
10
11 TreeNode weatherComparison(TreeNode _nodeA) : {
12   TreeNode root, nodeB;
13   Token t;
14 } {
15   <EQUALS> t = <WEATHER> {
16     root  = new TreeNodeEquals();
17     nodeB = new TreeNodeConstant(
18       ContextParameterWeather.stringToWeather(t.image));
19     root.setChilds(new TreeNode[]{ _nodeA, nodeB });
20     return root;
21   }
22 | <EQUALSNOT> t = <WEATHER> { ... }
23 }
```

Listing 4.1: Regeln der neuen Guard-Grammatik (Auszug).

Die beiden dargestellten Methoden definieren die gültige Syntax für eine Bedingung, welche den Kontextparameter *Wetter* verwendet. Dabei muss auf das Token `<CONTEXT_WEATHER>` ein Vergleichsoperator (`<EQUALS>` bzw. `<EQUALSNOT>`) sowie ein `<WEATHER>`-Token folgen. Darüber hinaus basiert

die Erstellung eines Verarbeitungsbaumes auf den Anweisungen dieser beiden Methoden, welche dadurch Objekte vom Typ `TreeNode` erzeugen bzw. als Rückgabeparameter zurückliefern. Die zur Erstellung der Datenstruktur notwendigen Klassen mussten dabei manuell erzeugt und mit den später für die Verarbeitung erforderlichen Funktionalitäten gefüllt werden. Gegensätzlich dazu wird von JavaCC mit Hilfe der erstellten Grammatikdefinition vollautomatisch ein dazu passender Parser erzeugt, welcher die Syntax von Eingabedaten (Transition-Guards bzw. kontextsensitive Bedingungen von Content-Objekten) prüft sowie mittels der im JavaCC-Dokument spezifizierten Java-Anweisungen einen die Eingabedaten repräsentierenden Verarbeitungsbaum erstellt. Abbildung 4.24 zeigt die vom Parser für die folgende Eingabe erzeugte interne Datenstruktur.

```
Context.Weather == SUNNY || Context.Weather == CLOUDY
```

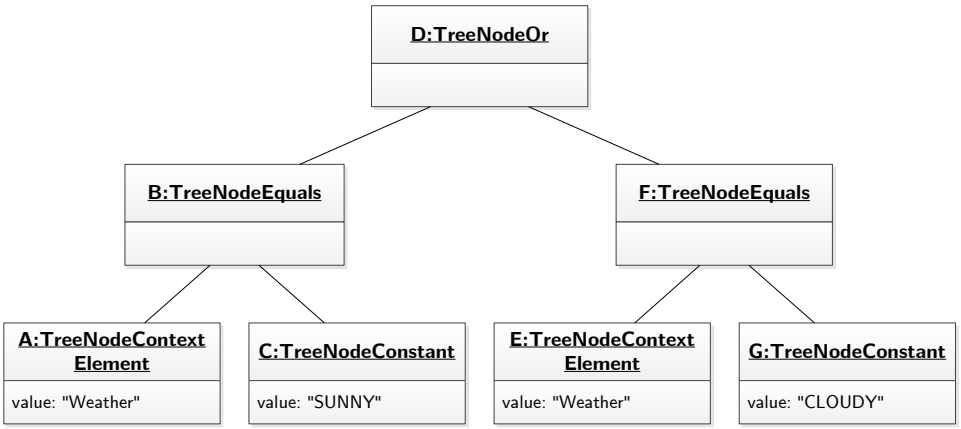


Abbildung 4.24: Vom JavaCC-Parser erstellte interne Baumstruktur.

Für alle in der Baumstruktur repräsentierten Informationen werden dafür vorgesehene Objekte angelegt. Durch eine in jeder dieser Klassen vorhandene – zur weiteren Verarbeitung der Eingabedaten implementierte – `evaluate()`-Methode ist es zur Laufzeit möglich, die Informationen der Datenstruktur auszuwerten und je nach Rückgabewert die mit der Bedingung assoziierte Transition durchzuführen bzw. zu ignorieren. Durch diesen Mechanismus werden einerseits die bereits angesprochenen Transition-Guards der Statechart-Logik, andererseits jedoch auch die mit diversen Inhaltsobjekten verknüpften Bedingungen geprüft. Somit ist es der Client-Anwendung möglich, je nach aktueller Kontextsituation die gewünschten Aktionen auszuführen bzw. Inhalte darzustellen.

4.4.3 XML-Schnittstelle

Die bereits existente XML-Schnittstelle dient der Übertragung relevanter Informationen vom Admin-Tool an das mobile Endgerät. Zu diesen Daten gehören einerseits alle im Statechart-Editor definierten Elemente – andererseits auch alle über diverse andere Module (Content-Integrator, Game-Area-Designer etc.) eingepflegten Informationen.

Für den Austausch von Daten zwischen zwei Systemen ist vor allem relevant, dass diese in einer bestimmten, global gültigen Form vorliegen. Die in Lomotain verwendeten Spielbeschreibungen basieren deshalb auf einer XML-Schema-Definition, deren Grundstruktur in Abbildung 4.25 dargestellt ist.

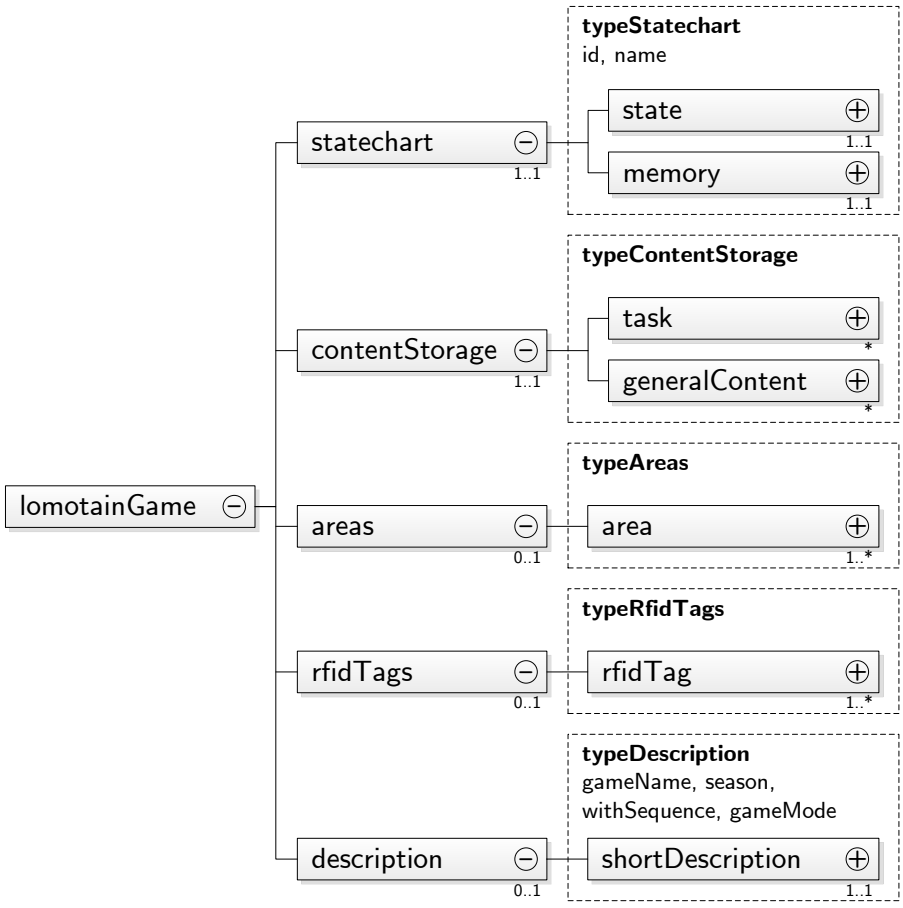


Abbildung 4.25: Grundstruktur der XML-Schema-Definition.

Das XML-Schema-Dokument teilt sich ausgehend vom Root-Element (`lomotainGame`) in fünf Bereiche:

- Der Abschnitt `statechart` inkludiert alle im Statechart-Editor erfassten Daten und somit die gesamte Anwendungslogik für das jeweilige

Spielszenario. Subelemente sind dabei die einzelnen States (mitsamt Transitions etc.) und benötigte interne Variablen.

- Das Element `contentStorage` beinhaltet Inhaltsdaten (Text, Bilder, Musik) – einerseits die der jeweiligen Tasks, andererseits auch global gültige bzw. verwendbare Informationen.
- Im `areas`-Abschnitt werden die für die jeweiligen Tasks spezifizierten Areale in Form von Koordinaten an die Client-Anwendung übermittelt.
- Ähnlich dazu verfügt der `rfidTags`-Bereich über alle im Spielszenario benötigten RFID-Tags inklusive Position und eindeutiger ID.
- Das Element `description` wird zur Übermittlung von allgemeinen Spielinformationen verwendet.

Für die Integration von kontextsensitiven Aspekten ist es u. a. notwendig, mehrere Inhaltsdefinitionen mitsamt Kontext-Bedingungen für ein einziges Inhaltsobjekt (z. B. Bild mit Aufgabenstellung) festzulegen. Hierfür wurde das XML-Schema-Dokument um die erforderlichen Konzepte erweitert. Abbildung 4.26 zeigt eines jener XML-Elemente, welche davon betroffen sind.

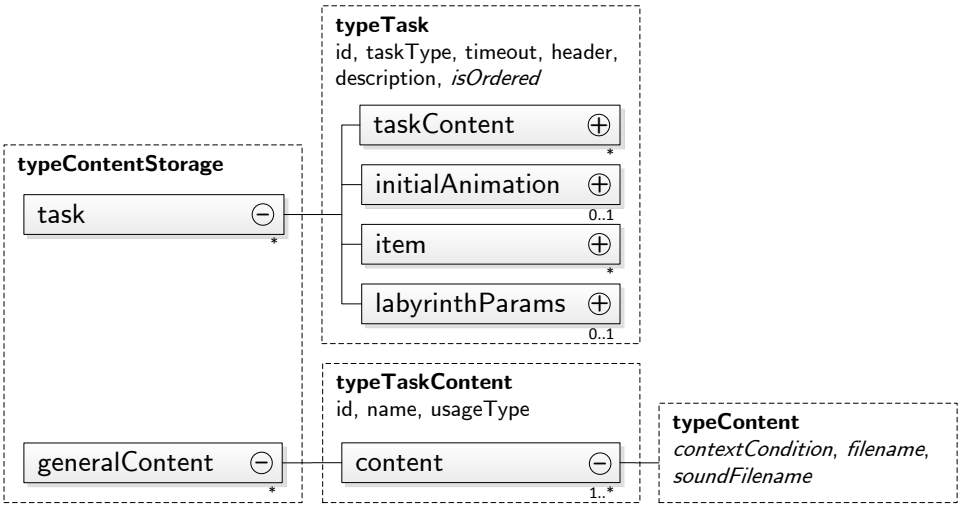


Abbildung 4.26: Definition kontextsensitiver Inhalte mittels XML-Schema.

Der Abschnitt `contentStorage` (Abbildung 4.25) besitzt u. a. eine beliebige Menge an `generalContent`-Subelementen, welche jeweils ein global verwendbares Inhaltsobjekt definieren. Im Gegensatz zur ursprünglichen Spezifikation erlaubt das XML-Schema nun, dass jedes dieser `generalContent`-Objekte beliebig viele (eines oder mehrere) `content`-Kindelemente enthält. Dieser Aspekt ist für das kontextsensitive Verhalten der Anwendung wichtig, da jedes `content`-Objekt – neben dem Inhalt selbst – optional auch

eine Kontext-Bedingung enthalten kann. Somit kann jedes `generalContent`-Objekt mehrere – für verschiedene Kontextsituationen gültige – Inhaltsdefinitionen beinhalten und so die kontextsensitiven Adaptionen ermöglichen.

Die soeben beschriebene Erweiterung der XML-Schema-Definition wird neben den globalen Inhaltsobjekten auch für die Definition von aufgabenspezifischen Inhalten verwendet und stellt die wichtigste Neuerung im Rahmen der XML-Schema-Spezifikation dar. Die im Statechart-Bereich verwendeten Transition-Guards waren bereits in der ursprünglichen XML-Schema-Definition vorhanden und werden als Textelemente abgelegt. Da hiermit die neuen, kontextsensitiven Bedingungen ebenfalls darstellbar sind, ist es nicht notwendig, in diesem Bereich Änderungen vorzunehmen.

4.5 Evaluierung

Nachdem die Implementierung der die kontextsensitiven Aspekte umsetzenden Erweiterungen vorgestellt wurde, soll ein Test-Szenario zeigen, dass die für Admin-Tool bzw. Client-Anwendung entwickelten Module die gewünschten Funktionalitäten erfüllen und mit Hilfe des vorgestellten Ansatzes die kontextsensitive Adaption sowohl von Anwendungslogik als auch von Inhalten innerhalb des Lomotain-Projekts ermöglichen. Ein mit den bereits in der Praxis verwendeten Adaptionskonzepten angestellter Vergleich soll außerdem zeigen, welche Vor- und Nachteile der auf Statecharts basierende Ansatz vor allem für Entwickler besitzt.

4.5.1 Test-Szenario

Das hier verwendete – für einen Themenpark durchaus denkbare – Spielszenario setzt sich aus mehreren Aufgaben zusammen, welche der Benutzer im Laufe der Spielausführung lösen soll. Folgende Regeln bilden die Basis für die Erstellung jener Anwendungslogik, welche später zur Ausführung des Spiels benötigt wird:

- Bei schönem Wetter (nicht regnerisch) sollen alle Benutzer *Aufgabe A* durchführen.
- Danach sollen alle Anwender unter 13 Jahren *Aufgabe B* ausführen – alle anderen *Aufgabe C*.
- Bei schlechtem Wetter (regnerisch) sollen alle Benutzer *Aufgabe D* durchführen. Für Anwender ab 13 Jahren soll diese Aufgabe eine kniffligere Fragestellung beinhalten.

Eine zur Definition dieses Szenarios passende Logikbeschreibung auf Basis von Statecharts könnte wie in Abbildung 4.27 dargestellt aussehen.

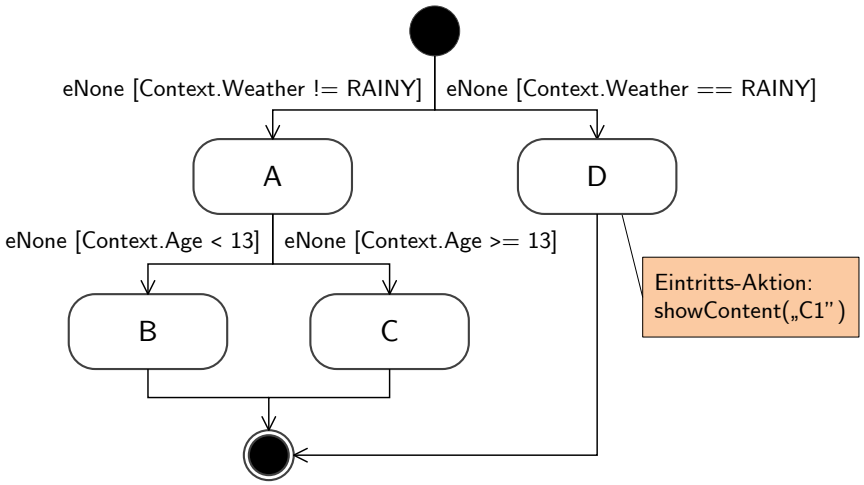


Abbildung 4.27: Logikdefinition des Test-Szenarios.

Das abgebildete Statechart zeigt das umgesetzte Szenario auf relativ abstrakter Ebene – auf die interne Logik der einzelnen Aufgaben wird in diesem Zusammenhang nicht näher eingegangen, da dies für die kontextsensitiven Aspekte nicht von Bedeutung ist. Jedoch ist ersichtlich, dass das in vier Aufgaben geteilte Szenario die benötigten Kontextparameter (Wetter und Alter) teils direkt in der Logik-Definition verwendet, teils jedoch im Rahmen verschiedener Content-Objekte – dargestellt in Tabelle 4.5 – einsetzt.

Content-Objekt	**Bedingung**	**Inhalt**
C1	`Context.Age < 13`	AufgabeD1.png
C1	`Context.Age >= 13`	AufgabeD2.png

Tabelle 4.5: Definition der Content-Objekte.

Die Ausführung der alle benötigten Informationen beinhaltenden Spielbeschreibung führt am Mobilgerät – beeinflusst von den gewählten Kontextparametern im Benutzerprofil – zu den gewünschten Ergebnissen, welche im Folgenden (Abbildungen 4.28, 4.29, 4.30, 4.31) dargestellt sind.

Die vier dargestellten Testfälle decken alle Möglichkeiten ab und zeigen, dass die definierten Aufgaben je nach aktueller Kontextsituation visualisiert werden. Dabei ist für den Anwender nicht ersichtlich, ob bzw. in welcher Form (Logik und/oder Inhalt) sich das vorliegende System an die aktuelle Situation adaptiert. Jedoch bestätigt das dargestellte Szenario, dass der in der vorliegenden Arbeit in Konzeption und Umsetzung beschriebene Ansatz auch in der Praxis ordnungsgemäß funktioniert und eingesetzt werden kann.

Abbildung 4.28: Testfall 1 (Alter: 24 Jahre, Wetter: Sonnig).

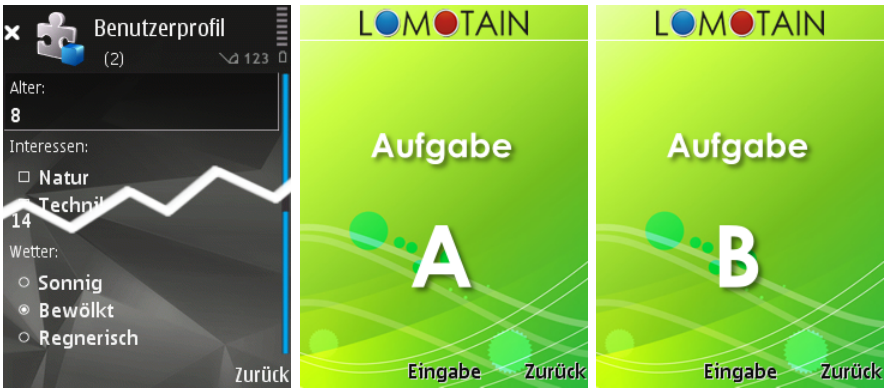


Abbildung 4.29: Testfall 2 (Alter: 8 Jahre, Wetter: Bewölkt).

4.5.2 Ergebnisse & Vergleich

Die Umsetzung des Test-Szenarios ist neben der hier vorgestellten Methode auch mit diversen weiteren Techniken (erläutert in Abschnitt 2.3.2) möglich, welche bei korrekter Verwendung jedoch alle zum gleichen Ergebnis – einer auf die Kontextsituation reagierenden Anwendung – führen. Aus den unterschiedlichen, in der Praxis bereits verwendeten Adaptionskonzepten lassen sich sowohl Vor- als auch Nachteile des im Rahmen dieser Diplomarbeit vorgestellten Verfahrens ableiten. Zu den wichtigsten Vorteilen des auf Statecharts basierenden Ansatzes gehört die Möglichkeit der grafischen Visualisierung. Im Gegensatz zum Großteil der anderen Adaptionskonzepte verwendet der hier entwickelte nicht nur textbasierte Werkzeuge, sondern außerdem Zustandsdiagramme, um die kontextsensitiven Aspekte zu definieren. Diese grafische Repräsentation erhöht die Lesbarkeit des definierten Systems (Anwendungslogik) und hilft dadurch vor allem Entwicklern, die logische Struktur des zu erstellenden Spielszenarios zu verstehen.

Abbildung 4.30: Testfall 3 (Alter: 8 Jahre, Wetter: Regnerisch).

Abbildung 4.31: Testfall 4 (Alter: 24 Jahre, Wetter: Regnerisch).

Ein Vorteil – besonders gegenüber dem Ansatz mit statischen Regeln – ist die Möglichkeit, jederzeit und relativ einfach die gewünschte Anwendungslogik mitsamt kontextsensitiven Bedingungen bzw. Aktionen austauschen zu können. Diese Flexibilität wird durch die Entkopplung des Anwendungsinhalts – definiert in der Spielbeschreibung – von der eigentlichen Anwendung am Mobilgerät ermöglicht.

Aufgrund der Verwendung von Statecharts ergibt sich ein weiterer Vorteil. Mit dem bereits weitgehend bewährten und hochentwickelten Konzept der Zustandsdiagramme lässt sich nahezu jegliche existierende Problemstellung lösen – dies kommt im Rahmen dieser Arbeit auch dem kontextsensitiven Verhalten der Anwendung zugute. Ebenfalls einen Vorteil im Zusammenhang mit Statecharts stellt die gute Tooling-Unterstützung dar – es existiert eine große Anzahl an Anwendungen und Frameworks (z. B. Eclipse Modeling Project), welche die Definition von Zustandsdiagrammen erlauben oder die Entwicklung darauf basierender Applikationen erleichtern.

Neben den soeben erläuterten Vorteilen gibt es auch Nachteile, mit welchen der auf Statecharts basierende Ansatz konfrontiert werden muss. Dazu zählen in erster Linie die relativ hohen Ansprüche an die mobilen Endgeräte, welche während der gesamten Entwicklungszeit deutlich wurden. Wichtigster Grund dafür ist die Verarbeitung der Statechart-Logik, welche auch die kontextsensitiven Aspekte beinhaltet und bei komplexen Szenarien relativ viel Rechenleistung benötigt. Einfache statische Regeln oder Filtermechanismen besitzen hier klare Vorteile.

Um die soeben identifizierten Vor- und Nachteile des im Rahmen dieser Arbeit vorgestellten Adaptionskonzepts verdeutlichen zu können, wird im Folgenden erläutert, wie das vorgestellte Test-Szenario mit Hilfe anderer Techniken umgesetzt werden könnte.

- Ein mit Hilfe von **Filtermechanismen** implementiertes kontextsensitives System würde etwa die im Benutzerprofil angegebenen Informationen verwenden, um jene Aufgaben zu selektieren, welche mit den aktuellen Kontextinformationen übereinstimmen. Nachteil dieses Verfahrens ist jedoch, dass der zur Laufzeit ausgeführte Filtervorgang meist nur textbasierte Ausdrücke entgegennimmt, welche – besonders bei umfangreichen Systemen – schwieriger zu verstehen sind als grafische Repräsentationen. Außerdem ist es unter Verwendung von Filtermechanismen nur unzureichend möglich, Adaptionen hinsichtlich Anwendungslogik durchzuführen.

- **Statische Regeln** werden direkt im Source-Code definiert und sind fester Bestandteil einer Anwendung. Neben der fehlenden grafischen Repräsentation sind solche Regeln außerdem meist nur mit relativ großem Aufwand austauschbar bzw. erweiterbar. Im Rahmen des Test-Szenarios würden dabei etwa `if`- oder `case`-Anweisungen definiert, welche die gewünschten Bedingungen darstellen, bestimmte Aktionen ausführen und so die Adaption der Anwendung an die gegebene Situation auslösen.

- Auch der Einsatz von **dynamischen Regeln** im Rahmen von kontextsensitiven Systemen ist möglich. Dabei werden z. B. in externen Dateien sogenannte ECA-Regeln (Event-Condition-Action-Regeln) definiert, welche von der Anwendung eingelesen und interpretiert werden. Die relativ einfach austauschbaren Regeln enthalten die nötigen Bedingungen und Aktionen, welche die Anwendung kontextsensitiv gestalten. Für das vorgestellte Test-Szenario würde dabei für jede gewünschte Situation eine Regel spezifiziert werden, welche als Aktion das Visualisieren der jeweiligen Aufgabe beinhaltet. Größter Nachteil dieser Technik ist wiederum der fehlende grafische Aspekt, da die ECA-Regeln meist in textueller Form vorliegen.

- **URL-Kodierung** nutzt neben einer Client-Anwendung einen entfernten Dienst (z. B. Webservice), um die benötigten kontextsensitiven Adaptionen durchzuführen. Bezogen auf das Test-Szenario würde dabei ein Server-Dienst über die aktuelle Situation (Alter, Wetter) benachrichtigt werden und im Gegenzug die dazu passenden Aufgaben liefern. Vorteil dieser Technik ist, dass ein Großteil der notwendigen Rechenoperationen von einem leistungsfähigen Server verrichtet wird – jedoch erfordert das Konzept eine ansonsten nicht benötigte Kommunikationsmöglichkeit. Außerdem ist es aufgrund der gegebenen, textbasierten Form von URLs relativ schwierig, die Informationen grafisch darzustellen.

- Ein auf verschiedene **Layer** basierender Ansatz teilt die Anwendung in unterschiedliche Bereiche, welche jeweils für eine bestimmte Kontextsituation verantwortlich sind. Die aktuellen Kontextinformationen bestimmen den gerade aktiven Layer, welcher z. B. im dargestellten Test-Szenario die jeweils darin definierten Aufgaben visualisiert. Nachteil dieser strikten Trennung ist ein gewisses Maß an Redundanz, falls verschiedene Layer teils gleiche Informationen beinhalten sollen.

- **Ontologien** können im Rahmen von kontextsensitiven Systemen ebenfalls zur Adaption verwendet werden. Dabei werden Relationen zwischen verschiedenen Objekten definiert, welche im späteren Verlauf dazu dienen, jene Objekte auszuwählen, welche ausgehend von einem Wurzelelement von Bedeutung sind. Bezogen auf das Test-Szenario könnte die Ontologie alle definierten Aufgaben, verknüpft mit Kontextsituationen, beinhalten. Ausgehend von der gerade herrschenden Situation ist es sodann möglich, die relevanten Aufgaben zu identifizieren und darzustellen. Dieser ebenfalls grafische Aspekte miteinbeziehende Ansatz ist jedoch – ähnlich wie Filtermechanismen – nicht optimal für die Adaption von Anwendungslogik geeignet.

Die soeben – im Bezug auf das Test-Szenario – vorgestellten, bisher verfügbaren Adaptionskonzepte zeigen, dass es keinen optimalen Ansatz gibt, welcher keinerlei Nachteile besitzt sowie keine Probleme mit sich bringt. Ebenfalls ersichtlich ist, dass die meisten bisher existierenden Techniken keine grafische Repräsentation der kontextsensitiven Regeln bzw. Bedingungen erlauben – dies stellt den großen Vorteil des auf Statecharts basierenden Konzepts dar. Hingegen besitzt dieser Ansatz auch Nachteile – etwa die hohe Verarbeitungskomplexität gegenüber einfacheren Filtermechanismen bzw. statischen Regeln. Ungeachtet dessen zeigt das vorgestellte Test-Szenario jedoch sehr deutlich, dass das in dieser Arbeit präsentierte Konzept zur kontextsensitiven Adaption von Anwendungen auch in der Praxis gut funktioniert, allen verlangten Anforderungen gerecht wird und nahezu jegliche situationsbedingte Adaption ermöglicht.

Kapitel 5

Resümee

5.1 Bewertung der Arbeit

Die vorliegende Diplomarbeit beschreibt einen bislang nicht existierenden, auf Statecharts beruhenden Ansatz zur kontextsensitiven Anwendungsadaption. Dabei wird bewiesen, dass dieses Konzept – neben bestehenden Adaptionsverfahren – ebenfalls verwendet werden kann, um Anwendungen kontextsensitiv zu gestalten. Ausgehend von diverse Ansätzen, um Kontextsensitivität in Statecharts zu integrieren, gelangt die Arbeit im Rahmen der Konzeptionierung zum Ergebnis, dass ein kombinierter Ansatz die passendste Möglichkeit hierfür darstellt. Neben der Möglichkeit, sowohl Anwendungs- bzw. Spiellogik als auch dargestellte Inhalte kontextsensitiv anpassen zu können, besitzt die hier beschriebene Technik die Besonderheit, die gewünschten kontextsensitiven Aspekte grafisch darstellen zu können. Aufgrund dieser Vorteile eignet sich dieses Verfahren hervorragend sowohl für das im Rahmen der Implementierung erweiterte Lomotain-System, als auch für jede weitere Applikation, deren interne Logik auf Statecharts beruht.

Die für Konzeptionierung und Umsetzung des neuen Adaptionsverfahrens notwendige Untersuchung von bereits bestehenden kontextsensitiven Anwendungen zeigt, dass die bisher existierenden bzw. in der Praxis eingesetzten Techniken zumeist auf Filtermechanismen oder Regeln zurückgreifen, um der jeweiligen Anwendung kontextsensitive Aspekte zu verleihen. Keines der vorgestellten Projekte verwendet hingegen Statecharts zur Anpassung der Applikation. Die besonderen Vorteile eines solchen auf Zustandsautomaten basierenden Verfahrens hinsichtlich Lesbarkeit, Komplexität und Flexibilität sowie die Einsatztauglichkeit des Konzepts in der Praxis belegt die anhand eines Test-Szenarios vorgenommene Evaluierung des neuen Adaptionsansatzes. Der angestellte Vergleich mit bereits existierenden Adaptionstechniken gelangt dabei des Weiteren zum Ergebnis, dass das hier entwickelte Konzept die geforderten Erwartungen optimal erfüllt, das Ziel der Arbeit – die Entwicklung kontextsensitiver Anwendungen mittels Statecharts – erreicht und

sich besonders aufgrund der hervorstechenden Eigenschaft – der grafischen Repräsentation der kontextsensitiven Aspekte – von bisherigen Lösungen zur kontextsensitiven Anwendungsadaption klar unterscheidet.

5.2 Ausblick

Das Hauptziel der Arbeit – die Entwicklung eines auf Statecharts basierenden Verfahrens zur kontextsensitiven Adaption von Applikationen – konnte erfolgreich umgesetzt werden. Im Laufe der Entwicklungszeit bzw. im Rahmen der Vorprojekte zu dieser Diplomarbeit kristallisierten sich jedoch einige weitere Ziele heraus, welche zwar für die vorliegende Arbeit nicht relevant sind – für das Gesamtprojekt Lomotain jedoch wichtige Aspekte darstellen:

- **Feldtest:** Ein geplanter, jedoch bisweilen noch nicht durchgeführter Feldtest wäre ein erster Versuch, künftige Benutzer mit dem System zu konfrontieren. Bei geeigneter Durchführung kann dabei eine Reihe interessanter Fragestellungen beantwortet werden – etwa in Hinblick auf Benutzerfreundlichkeit und Systemverhalten.

- **GUI:** Die derzeit mit Hilfe von Java ME-Bordmitteln umgesetzte Benutzeroberfläche der Client-Anwendung könnte unter Verwendung geeigneter Technologien (z. B. JavaFX) in neuem Glanz erstrahlen und somit der Anwendung mehr Professionalität bezogen auf Visualisierungsaspekte verleihen.

- **Update-Mechanismus:** Eine speziell für den kontextsensitiven Bereich der Anwendung relevante Erweiterung stellt ein Mechanismus zur Aktualisierung der Kontextinformationen dar. Die aktuell per Benutzerprofil im Vorfeld des Spiels angegebenen, relativ statisch gehaltenen Kontextdaten könnten um dynamische Aspekte ergänzt werden – etwa durch die Abfrage webbasierter Wetterdienste. Ein geeigneter Update-Mechanismus wäre dabei nötig, um auch zur Laufzeit eines Spielszenarios auf Änderungen der Kontextsituation reagieren zu können.

- **Statechart-Konzept:** Die Statechart-Komponente, welche nun neben der Logik auch kontextsensitive Aspekte beinhaltet, besitzt ebenfalls Verbesserungspotenzial. Während der bisherigen Entwicklung hat sich erwiesen, dass – besonders unter Verwendung aktuell verfügbarer Mobiltelefone mit RFID-Technologie (Nokia 6131, Nokia 6212) – Performanceprobleme auftreten. Dies hat u. a. den Grund, dass die am Client benötigte Statechart-Logik dort relativ unperformant verarbeitet werden kann. Eine mögliche, jedoch tief ins System eingreifende Lösung wäre bspw. die Einführung eines neuen Ansatzes. Anstatt die definierte Anwendungslogik in Form von Statecharts an den Client zu übertragen, könnten daraus im Admin-Tool bereits lauffähige Java-Klassen

generiert werden, welche die mobile Anwendung später einbindet und verwendet.

- **Android:** Um die Client-Anwendung für weitere Plattformen zur Verfügung stellen zu können, wäre die Portierung der Applikation hinsichtlich Lauffähigkeit auf anderen mobilen Betriebssystemen ein möglicher Ansatz. Dabei bietet sich etwa die moderne Android[1]-Plattform an, deren Verbreitung derzeit stetig zunimmt. Die Verwendung solch moderner Plattformen würde als besonderen Vorteil den Einsatz neuester Mobilgeräte ermöglichen, welche i. d. R. mit großzügiger Hardwareausstattung geliefert werden. Dies würde für die gesamte Lomotain Client-Anwendung eine Performancesteigerung bedeuten.

Aufgrund der vom aktuellen System zur Verfügung gestellten Funktionalitäten und einigen interessanten Kontakten aus der Wirtschaft ist aus aktueller Sicht die Fortführung und Weiterentwicklung von Lomotain – in Form eines Gesamtprojekts oder als einzelne Teilprojekte – auf jeden Fall wünschenswert, zumal es noch viele lukrative Aufgabenstellungen zu bearbeiten gilt und teils hohes Interesse an Kooperationen mit Unternehmen aus dem Tourismussektor existiert.

[1] http://www.android.com

Anhang A

Anhang

Die folgenden Tabellen zeigen die detaillierte Auswertung der in Kapitel 2 untersuchten Anwendungen. Ausgefüllte Kreise (•) bedeuten dabei eine explizite Verwendung des jeweiligen Konzepts – leere Kreise (◦) nur die prinzipielle Einsatzmöglichkeit im Rahmen des Projekts.

	Position	**Benutzerinteressen**	**Uhrzeit**	**Geräteeigenschaften**	**Geschwindigkeit**	**Netzwerk**	**Alter**	**Geschlecht**	**Kompass**	**Körperwerte**	**Temperatur**
ParcTab	•		•								
C-MAP		•									
Conference Assistant		•									
GUIDE	•	•	•								
News Dude		•									
CybreMinder	•		•								•
ARREAL	•				•						
CRUMPET	•	•		•		•					
HyperAudio	•										
HIPS	•	•									
PinPoint	•		•	•							
Gulliver's Genie	•	•		•							
LOL@	•										

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

	Position	**Benutzerinteressen**	**Uhrzeit**	**Geräteeigenschaften**	**Geschwindigkeit**	**Netzwerk**	**Alter**	**Geschlecht**	**Kompass**	**Körperwerte**	**Temperatur**
mobiDENK	•	•									
Sightseeing4U	•	•		•							
COMPASS	•	•	•								
m-ToGuide	•	•	•			•					
xPOI	•		•		•						
Chameleon				•		•					
MOPET	•				•		•	•		•	
Ubidoo	•		•								
Conny	•	•					•	•			
Tacticycle	•								•		

Tabelle A.1: Verwendete Kontextparameter.

	Informations-gehalt	**Darstellung**	**Verhalten**
ParcTab	•		
C-MAP	•		
Conference Assistant	•		
GUIDE	•	•	
News Dude	•		
CybreMinder	•	•	
ARREAL		•	
CRUMPET	•		
HyperAudio	•	•	
HIPS	•	•	
PinPoint	•	•	
Chisel	○	○	○
Gulliver's Genie	•	•	
LOL@	•	•	
mobiDENK	•	•	

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

	Informations-gehalt	**Darstellung**	**Verhalten**
Sightseeing4U	•		
CASS	○	○	○
COMPASS	•		
m-ToGuide	•	•	
xPOI		•	
Chameleon		•	
ContextL			•
MOPET			•
Ubidoo	•		
Conny			•
Tacticycle	•		

Tabelle A.2: Verwendete Adaptionsarten.

	Filtermechanismen	**Dynamische Regeln**	**Statische Regeln**	**URL-Kodierung**	**Benutzerfeedback**	**Layer**	**Ontologie**
ParcTab	•						
C-MAP	•						
Conference Assistant	•						
GUIDE				•			
News Dude	•				•		
CybreMinder		•					
ARREAL			•				
CRUMPET	•						
HyperAudio		•					
HIPS		•					
PinPoint				•			
Chisel		•					
Gulliver's Genie	•						

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

	Filtermechanismen	**Dynamische Regeln**	**Statische Regeln**	**URL-Kodierung**	**Benutzerfeedback**	**Layer**	**Ontologie**
LOL@				•			
mobiDENK	•						
Sightseeing4U	•						
CASS		•					
COMPASS	•						
m-ToGuide	•						
xPOI		•					
Chameleon					•		
ContextL						•	
MOPET			•				
Ubidoo							•
Conny		•					
Tacticycle			•				

Tabelle A.3: Verwendete Adaptionskonzepte.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Abowd, G. D., A. K. Dey, P. J. Brown, N. Davies, M. Smith und P. Steggles: *Towards a Better Understanding of Context and Context-Awareness.* In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99)*, S. 304–307, Karlsruhe, Deutschland, Juni 1999.

[2] Baldauf, M., S. Dustdar und F. Rosenberg: *A Survey on Context-Aware Systems.* International Journal of Ad Hoc and Ubiquitous Computing, 2(4):263–277, Juni 2007.

[3] Baldzer, J., S. Boll, P. Klante, J. Krösche, J. Meyer, N. Rump, A. Scherp und H. J. Appelrath: *Location-Aware Mobile Multimedia Applications on the Niccimon Platform.* In: Gesamtzentrum für Verkehr Braunschweig e. V. (GZVB) (Hrsg.): *2. Braunschweiger Symposium Informationssysteme für mobile Anwendungen (IMA '04)*, S. 318–334, Braunschweig, Deutschland, Oktober 2004.

[4] Bauer, J.: *Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic.* Diplomarbeit, Technische Universität Berlin, März 2003.

[5] Beer, W., V. Christian, A. Ferscha und L. Mehrmann: *Modeling Context-Aware Behavior by Interpreted ECA Rules.* In: *Proceedings of the International Conference on Parallel and Distributed Computing (EuroPar '03)*, S. 26–29, Klagenfurt, Österreich, August 2003.

[6] B'Far, R.: *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML.* Cambridge University Press, Cambridge, England, 2004.

[7] Billsus, D. und M. J. Pazzani: *A Personal News Agent that Talks, Learns and Explains.* In: *Proceedings of the 3rd Annual Conference on Autonomous Agents (AGENTS '99)*, S. 268–275, Seattle, Washington, USA, Mai 1999.

[8] Boll, S. C. J.: *Multmedia Sightseeing 4 U – What Web Services Can Do for Personalized Multimedia Applications.* In: *Proceedings of the*

7th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI '03), Bd. XVI, S. 220–225, Orlando, Florida, USA, Juli 2003.

[9] Buchholz, T., A. Küpper und M. Schiffers: *Quality of Context: What It Is And Why We Need It.* In: *Proceedings of the 10th International Workshop of the HP OpenView University Association (HP-OVUA '03)*, Genf, Schweiz, Juli 2003.

[10] Buttussi, F. und L. Chittaro: *MOPET: A Context-Aware and User-Adaptive Wearable System for Fitness Training.* Artificial Intelligence in Medicine, 42(2):153–163, Februar 2008.

[11] Chen, G. und D. Kotz: *A Survey of Context-Aware Mobile Computing Research.* Techn. Ber. TR2000-381, Department of Computer Science, Dartmouth College, Hanover, New Hampsihre, USA, November 2000.

[12] Cheverst, K., N. Davies, K. Mitchell und A. Friday: *The Role of Connectivity in Supporting Context-Sensitive Applications.* In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99)*, S. 193–207, Karlsruhe, Deutschland, September 1999.

[13] Copeland, T.: *Generating Parsers with JavaCC.* Centennial Books, Alexandria, Virginia, USA, 2007.

[14] Costanza, P., R. Hirschfeld und W. D. Meuter: *Efficient Layer Activation for Switching Context-dependent Behavior.* In: *Proceedings of the Joint Modular Languages Conference (JMLC '06)*, S. 84–103, Oxford, England, September 2006.

[15] Desmet, B., J. Vallejos, P. Costanza und R. Hirschfeld: *Layered Design Approach for Context-Aware Systems.* In: *Proceedings of the 1st International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '07)*, S. 157–165, Limerick, Irland, Januar 2007.

[16] Dey, A. K. und G. D. Abowd: *CybreMinder: A Context-Aware System for Supporting Reminders.* In: *Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC '00)*, S. 172–186, Bristol, England, September 2000.

[17] Dey, A. K., D. Salber, G. D. Abowd und M. Futakawa: *An Architecture To Support Context-Aware Applications.* Techn. Ber. GIT-GVU-99-23, Georgia Institute of Technology, Juni 1999.

[18] Dey, A. K., D. Salber, G. D. Abowd und M. Futakawa: *The Conference Assistant: Combining Context-Awareness with Wearable Computing.* In: *Proceedings of the 3rd IEEE International Symposium on Wearable Computers (ISWC '99)*, S. 21–28, San Francisco, Kalifornien, USA, Oktober 1999.

[19] Dürr, F., J. Palauro, L. Geiger, R. Lange und K. Rothermel: *Ein kontextbezogener Instant-Messaging-Dienst auf Basis des XMPP-Protokolls.* In: *5. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste*, Bd. 42 d. Reihe *Sonderdruck Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg*, S. 23–28, Nürnberg, Deutschland, September 2008.

[20] Fahy, P. und S. Clarke: *CASS – Middleware for Mobile Context-Aware Applications.* In: *Proceedings of the Workshop on Context Awareness, 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys '04)*, Boston, Massachusetts, USA, Juni 2004.

[21] Fels, S., Y. Sumi, T. Etani, N. Simonet, K. Kobayashi und K. Mase: *Progress of C-MAP: A Context-aware Mobile Assistant.* In: *Proceedings of AAAI 1998 Spring Symposium on Intelligent Environments*, S. 60–67, Paolo Alto, Kalifornien, USA, März 1998.

[22] FH OÖ Studienbetriebs GmbH: *Location Based Mobile Gaming and Entertainment – LOMOTAIN: Abschlussbericht*, 2008.

[23] Fuchs, F.: *Kontextsensitive Dienste.* Technische Universität München, Fakultät für Informatik, Januar 2002. http://www.contextsensitive.de, Abruf: 22.02.2010.

[24] Greenberg, J.: *Metadata and the World Wide Web.* Encyclopedia of Library and Information Science, 3:1876–1888, Juni 2003.

[25] Gronback, R. C.: *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley, Boston, Massachusetts, USA, 2009.

[26] Harel, D.: *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming, 8(3):231–274, Juni 1987.

[27] Harold, E. R. und W. S. Means: *XML in a Nutshell.* O'Reilly, Sebastopol, Kalifornien, USA, 2002.

[28] Henricksen, K., J. Indulska und A. Rakotonirainy: *Generating Context Management Infrastructure from High-Level Context Models.* In: *Proceedings of the 4th International Conference on Mobile Data Management (MDM '03)*, S. 1–6, Melbourne, Australien, Januar 2003.

[29] Hofer, T., W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann und W. Retschitzegger: *Context-Awareness on Mobile Devices – The Hydrogen Approach.* In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS '03) - Track 9*, Bd. 9, S. 292, Big Island, Hawaii, USA, Januar 2003.

[30] Keeney, J. und V. Cahill: *Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework.* In: *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '03)*, S. 3–14, Como, Italien, Juni 2003.

[31] Korpipää, P. und J. Mäntyjärvi: *An Ontology for Mobile Device Sensor-Based Context Awareness.* In: *Proceedings of the 4th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT '03)*, S. 451–458, Stanford, Kalifornien, USA, Juni 2003.

[32] Krösche, J., J. Baldzer und S. Boll: *MobiDENK – Mobile Multimedia in Monument Conservation.* IEEE MultiMedia, 11(2):72–77, 2004.

[33] Krösche, J. und S. Boll: *The xPOI Concept.* In: Strang, T. und C. Linnhoff-Popien (Hrsg.): *Proceedings of the 1st International Workshop on Location and Context Awareness (LoCA '05)*, S. 113–119, Oberpfaffenhofen, Deutschland, Mai 2005.

[34] McAffer, J. und J. M. Lemieux: *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications.* Addison-Wesley, Boston, Massachusetts, USA, 2006.

[35] McCarthy, J. und S. Buvac: *Formalizing Context (Expanded Notes).* Techn. Ber. CS-TN-94-13, Stanford University, Stanford, Kalifornien, USA, Oktober 1994.

[36] Miraoui, M., C. Tadj und C. ben Amar: *Context Modeling and Context-Aware Service Adaptation for Pervasive Computing Systems.* International Journal of Computer and Information Science and Engineering (IJCISE), 2(3):148–157, 2008.

[37] Mohomed, I., J. C. Cai und E. de Lara: *URICA: Usage-awaRe Interactive Content Adaptation for Mobile Devices.* ACM SIGOPS Operating Systems Review, 40(4):345–358, Oktober 2006.

[38] Nicklas, D., M. Großmann, T. Schwarz, S. Volz und B. Mitschang: *A Model-Based, Open Architecture for Mobile, Spatially Aware Applications.* In: *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD '01)*, S. 117–135, Redondo Beach, Kalifornien, USA, Juli 2001.

[39] Object Management Group: *Unified Modeling Language 2.2 – Superstructure Specification*, Februar 2009. http://www.omg.org/spec/UML/2.2/Superstructure/PDF, Abruf: 01.03.2010.

[40] O'Hare, G. M. P. und M. J. O'Grady: *Gulliver's Genie: A Multi-Agent System for Ubiquitous and Intelligent Content Delivery.* Computer Communications, 26:1177–1187, 2003.

[41] Öztürk, P. und A. Aamodt: *Towards a Model of Context for Case-Based Diagnostic Problem Solving.* In: *Proceedings of the Interdisciplinary Conference on Modeling and Using Context (CONTEXT '97)*, S. 198–208, Rio de Janeiro, Brasilien, Februar 1997.

[42] Petrelli, D., E. Not, M. Zancanaro, C. Strapparava und O. Stock: *Modelling and Adapting to Context.* Personal and Ubiquitous Computing, 5(1):20–24, Februar 2001.

[43] Poppinga, B., M. Pielot und S. Boll: *Tacticycle: A Tactile Display for Supporting Tourists on a Bicycle Trip.* In: *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '09)*, S. 1–4, Bonn, Deutschland, September 2009.

[44] Poslad, S., H. Laamanen, R. Malaka, A. Nick, P. Buckle und A. Zipl: *CRUMPET: Creation of User-Friendly Mobile Services Personalised for Tourism.* In: *Proceedings of the 2nd International Conference on 3G Mobile Communication Technologies*, S. 28–32, London, England, März 2001.

[45] Roth, J.: *Context-Aware Web Applications Using the PinPoint Infrastructure.* In: *International Conference WWW/Internet (IADIS '02)*, S. 13–15, Lissabon, Portugal, November 2002.

[46] Samek, M.: *Practical Statecharts in C/C++: Quantum Programming for Embedded Systems.* CMP Books, Lawrence, Kansas, USA, Juli 2002.

[47] Samulowitz, M., F. Michahelles und C. Linnhoff-Popien: *CAPEUS: An Architecture for Context-Aware Selection and Execution of Services.* In: *Proceedings of the 3rd IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS '01)*, S. 23–39, Krakau, Polen, September 2001.

[48] Scheuchenegger, M.: *Entwurf und Implementierung eines Statechart-Interpreters für generische Anwendungslogik am Beispiel eines mobilen ortsbezogenen Spiels.* Diplomarbeit, Fachhochschule Oberösterreich – Studiengang Mobile Computing, August 2008.

[49] Schilit, B., N. Adams und R. Want: *Context-Aware Computing Applications.* In: *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '94)*, S. 85–90, Santa Cruz, Kalifornien, USA, Dezember 1994.

[50] Schwinger, W., C. Grün, B. Pröll, W. Retschitzegger und A. Schauerhuber: *Context-Awareness in Mobile Tourism Guides – A Comprehensive Survey.* In: Khalil-Ibrahim, I. (Hrsg.): *Handbook of Research on Mobile*

Multimedia, Second Edition, S. 298–314. Information Science Reference, 2008.

[51] Setten, M. van, S. Pokraev und J. Koolwaaij: *Context-Aware Recommendations in the Mobile Tourist Application COMPASS*. In: Bra, P. D. und W. Nejdl (Hrsg.): *Proceedings of the 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH '04)*, S. 235–244, Eindhoven, Niederlande, August 2004.

[52] Shaker, P. und N. T. S.: *The Java Parsing System*. In: *Proceedings of the Newfoundland Electrical and Computer Engineering Conference (NECEC '04)*, St. John's, Newfoundland, Kanada, November 2004.

[53] Sheng, Q. Z. und B. Benatallah: *ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development*. In: *Proceedings of the International Conference on Mobile Business (ICMB '05)*, S. 206–212, Sydney, Australien, Juli 2005.

[54] Stahl, C. und D. Heckmann: *Using Semantic Web Technology for Ubiquitous Location and Situation Modeling*. The Journal of Geographic Information Sciences CPGIS, 10(2):157–165, Dezember 2004.

[55] Stahl, C., D. Heckmann, T. Schwartz und O. Fickert: *Here and Now: A User-Adaptive and Location-Aware Task Planner*. In: *Proceedings of the International Workshop on Ubiquitous and Decentralized User Modeling (UbiDeUM '07)*, S. 52–63, Korfu, Griechenland, Juni 2007.

[56] Steinberg, D., F. Budinsky, M. Paternostro und E. Merks: *EMF: Eclipse Modeling Framework*. Addison-Wesley, Boston, Massachusetts, USA, 2009.

[57] Strang, T. und C. Linnhoff-Popien: *A Context Modeling Survey*. In: *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management, 6th International Conference on Ubiquitous Computing (UbiComp '04)*, S. 34–41, Nottingham, England, September 2004.

[58] Ullenboom, C.: *Java ist auch eine Insel: Programmieren mit der Java Standard Edition Version 6*. Galileo Press, Bonn, Deutschland, 8. Aufl., Januar 2009.

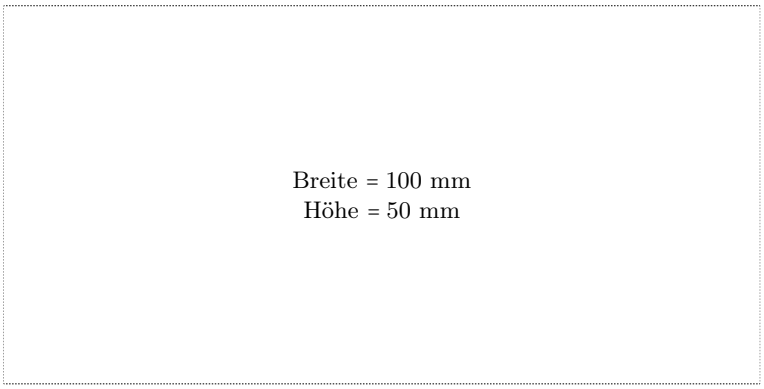
[59] Umlauft, M., G. Pospischil, Günther Niklfeld und E. Michlmayr: *LOL@, A Mobile Tourist Guide for UMTS*. Information Technology & Tourism, 5(3):151–164, 2003.

[60] Wahlster, W., J. Baus, C. Kray und A. Krüger: *REAL: Ein ressourcenadaptierendes mobiles Navigationssystem*. Informatik – Forschung und Entwicklung, Themenheft Mobile Computing, 16(4):233–241, November 2001.

[61] Want, R., B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis und M. Weiser: *An Overview of the ParcTab Ubiquitous Computing Experiment.* IEEE Personal Communications, 2(6):28–43, Dezember 1995.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —