Karlsruhe Reports in Informatics 2013,12
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Ubiquitäre Systeme (Seminar) SS 2013

Mobile und Verteilte Systeme
Ubiquitous Computing

Teil IX

Herausgeber:
Predrag Jakimovski, Martin Alexander Neumann,
Matthias Budde, Nadezda Sackmann

2013

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Karlsruhe Institute of Technology

Ubiquitäre Systeme (Seminar)
SS 2013

Mobile und Verteilte Systeme
Ubiquitous Computing

Teil IX

Herausgeber

Predrag Jakimovski
Martin Alexander Neumann
Matthias Budde
Nadezda Sackmann

Karlsruhe Institute of Technology (KIT)
Fakultät für Informatik
Lehrstuhl für Pervasive Computing Systems (PCS) und TECO

Interner Bericht 2013 - 12

Vorwort

Die Seminarreihe Ubiquitäre Informationssysteme hat eine lange Tradition in der Forschungsgruppe TecO. Ziel der Seminarreihe ist die Aufarbeitung und Diskussion aktueller Forschungsfragen im Bereich Ubiquitous Computing. Seit dem Wintersemester 2003/2004 werden die Seminararbeiten als KIT-Berichte veröffentlicht. Seit das TecO im Wintersemester 2010/2011 Teil der Forschungsgruppe Pervasive Computing Systems wurde, findet das Seminar nun in jedem Semester statt. Dieser Seminarband fasst die Ergebnisse der Arbeiten des Sommersemesters 2013 zusammen. Die Themenvielfalt der hier zusammengetragenen Aufsätze reicht von Kategorisierung und Softwareentwicklung für mobile B2B-Anwendungen, kontextsensitive Anwendungen, bis hin zu personalisierte und verteilte Sensornetze im Bereich Air Quality Sensing.

Hiermit danken wir den Studierenden für ihren besonderen Einsatz sowohl während des Seminars als auch bei der Fertigstellung dieses Bandes.

Karlsruhe, den 21. Oktober 2013

Predrag Jakimovski
Martin Alexander Neumann
Matthias Budde
Nadezda Sackmann

Inhaltsverzeichnis

Die Entwicklung von Kontextsensitiven Anwendungen

Ubiquitäre Systeme Seminar SS2013

Jannik Quehl[1] und Martin Alexander Neumann[2]

Karlsruhe Institute of Technology (KIT), Pervasive Computing Systems - TECO
[1]jannik.quehl@student.kit.edu
[2]mneumann@teco.edu

Zusammenfassung Kontextsensistive Systeme beschreiben eine Klasse von Anwendungen und Services, die abhängig von ihrer Umgebung und von Eigenschaften des Benutzers ihr Verhalten ändern können. Solche Systeme finden immer stärkere Verbreitung, insbesonders aufgrund der stetig wachsenden Verteilung von Smartphones und ähnlichen mobilen Geräten. Diese Ausarbeitung setzt sich damit auseinander, wie sich Kontextdaten allgemein modellieren lassen und welche Möglichkeiten es darauf aufbauend gibt, solche Anwendungen zu entwickeln.

1 Einleitung

1.1 Motivation

Wenn Menschen miteinander interagieren, hängt ihr genaues Verhalten von vielen Faktoren ab. Menschen interagieren zum Beispiel mit fremden Personen anders als mit Freunden, sie sprechen lauter wenn die Umgebungsgeräusche laut sind und das Verhalten zwischen denselben Menschen kann sich vollständig ändern wenn weitere Personen hinzukommen.

Computer auf der anderen Seite benutzen in der Regel weit weniger Informationen über ihre Umwelt, um ihr Verhalten zu bestimmen. Die meisten Programme und Anwendungen reagieren ausschließlich auf das, was der Benutzer eingibt und liefern für dieselbe Benutzereingabe immer dasselbe Ergebnis. Um die Interaktion zwischen Mensch und Computer zu verbessern und für den Menschen intuitiver zu gestalten, bietet es sich daher an, bei der Entwicklung von Programmen und Anwendungen zusätzlich zu den direkten Benutzereingaben weitere Informationen, wie die Uhrzeit, Vorhandensein einer Internetverbindung oder Identität des aktuellen Benutzers des Systems in das Verhalten der Anwendung einfließen zu lassen.

Insbesondere bei der Entwicklung von Anwendungen für moderne mobile Geräte, wie Laptops, Smartphones oder Tablets, bei denen sich die für die Benutzung relevanten Informationen während des Ausführens häufig ändern und bei denen die Anzahl der relevanten Daten durch die Vielzahl der Sensoren des

Gerätes sehr hoch sein können, ist es sinnvoll, diese Daten in einem gemeinsamen Modell zusammenzufassen und einheitlich zu verwalten, um zu vermeiden, dass für jeden Sensor ein neues Modell innerhalb der Anwendung erdacht werden muss.

Die Gesamtheit dieser relevanten Informationen fasst man unter dem Begriff Kontext zusammen. Anwendungen, die per Sensoren ihren Kontext bestimmen und ihn benutzen, um ihr Verhalten an die aktuelle Situation anzupassen, nennt man kontextsensitiv.

Diese Arbeit befasst sich damit, wie solche Kontexte modelliert werden können und darauf aufbauend, wie kontextsensitive Anwendungen entworfen und umgesetzt werden können. Hierzu werden zuerst einmal in den folgenden Abschnitten die Begrifflichkeiten des Kontexts und der Kontextsensitivität definiert. Anschließend werden in Abschnitt 2 die Grundlegenden Modelle zur Modellierung des Kontextes, sowie die allgemeinen möglichen Architekturen von kontextsensitiven Systemen eingeführt. Abschnitt 3 befasst sich damit, wie in der Praxis verschiedene Modellierungsansätze umgesetzt werden, mit besonderem Augenmerk auf ontologiebasierte, logikbasierte und modellgetriebene Ansätze. Abschnitt 4 beinhaltet die Diskussion der einzelnen Ansätze mit dem Ziel, herauszuarbeiten, unter welchen Gegebenheiten die einzelnen Ansätze zur Entwicklung eines kontextsensitiven Systems am besten zu verwenden sind, sowie die Konklusion dieser Arbeit.

1.2 Definition: Kontext

Um über Kontext und kontextsensitive Systeme sprechen zu können, müssen diese Begriffe zunächst einmal definiert werden. Eine erste, frühe formale Definition des Begriffes Kontext lieferten Schilit et al., indem sie Kontext als Ort, Identität naher Personen und Objekte, sowie Änderungen an diesen Objekten bezeichneten. [22] Ähnliche Definitionen von Kontext nehmen weitere Kenngrößen zu dieser Definition, wie zum Beispiel Tageszeit, Datum, Temperatur[5] oder sogar den emotionalen Zustand des Benutzers und den Fokus seiner Aufmerksamkeit[13] hinzu, um den Begriff Kontext möglichst weit zu fassen.

In vielen Definitionen wird zudem zwischen verschiedenen Klassen von Kontexten unterschieden. Chen und Kotz schlagen so zum Beispiel vor Schilits Definition zu erweitern und Kontextdaten einer Anwendung in vier Kategorien von Kontexten einzuteilen[6]:

Rechenkontext
Daten, die den aktuellen Zustand des Gerätes beschreiben und für die Leistung des Gerätes relevant sein können, wie Netzwerkverbindung, Kommunikationsbandbreite...

Benutzerkontext
Daten, die den Benutzer und sein Umfeld auf Persönlicher und/oder sozialer Ebene beschreiben, wie Benutzerprofil, Nähe anderer Personen...

Physischer Kontext
Daten, die die physischen Begebenheiten rund um das Gerät und den Benutzer beschreiben, wie Belichtung, Umgebungslautstärke, Temperatur...

Zeitkontext
Daten, die den aktuellen Zeitpunkt in unterschiedlicher Genauigkeit festlegen, wie Uhrzeit, Datum, Kalenderwoche...

Im Kontext dieser Seminararbeit ist eine solch strikte Einteilung jedoch nicht vonnöten, da die Kontextmodelle der einzelnen Ansätze im Allgemeinen keine solche Trennung vorsehen, weswegen eine allgemeinere Definition ausreicht. Eine solche Definition findet sich zum Beispiel bei Dey und Abowd in „*Towards a Better Unterstanding of Context and Context-Awareness*"[2] worin sie als Definition vorschlagen:

> „*Der Begriff Kontext bezeichnet alle Informationen, die benutzt werden können, um die Situation einer Entität zu beschreiben. Eine Entität ist hierbei eine Person, ein Ort oder ein Objekt das relevant für die Interaktion zwischen dem Benutzer und einer Anwendung gehalten wird, inklusive dem Benutzer und der Anwendung selber.*"

1.3 Definition: Kontextsensitivität

Programme und Anwendungen, die ihren Kontext benutzen, um ihr Verhalten anzupassen, nennt man im allgemeinen kontextsensitiv. Es gibt jedoch unterschiedliche Auffassungen, wie man die Kontextsensitivität formal am besten definiert. Eine frühe Definition geben zum Beispiel Schilit et al., die in ihrer sehr allgemein gehaltenen Definition, alle Anwendungen als kontextsensitiv bezeichnen, die sich selber an ihren Kontext adaptieren.[22] Hierbei unterscheiden sie auch zwischen verschiedenen Formen der Adaption, wie eine automatisch Auswahl von Daten die dem Kontext ähneln (z.B. Objekte in der direkten Umgebung), dem Anpassen einzelner Befehle in der Benutzerschnittstelle (z.B. Aktivieren der Möglichkeit mit Personen der Umgebung zu interagieren) oder dem Abstimmen der ganzen Anwendung auf den jeweiligen Kontext (z.B. Verwendung von dunkleren Farben in der GUI falls die Umgebung dunkel ist).

Viele andere Ansätze, den Begriff der Kontextsensitivität genau zu definieren lassen sich gut in zwei Klassen von einteilen:

- Kontextsensitivität im Sinne des Detektieren und/oder Verarbeiten des Kontextes auf eine beliebige, nicht weitere spezifizierte Art und Weise
- Kontextsensitivität im Sinne von konkreten Handlungsoptionen, die vom aktuellen Kontext abhängen

Als Beispiel für den allgemeineren Fall der Kontextnutzung, definieren Hull et al. Kontextsensitivität als die Fähigkeit, Aspekte des lokalen Umfelds des Nutzers zu erfassen und wahrzunehmen, sie zu interpretieren und auf sie reagieren zu

können[19]. Beispiel für den spezielleren Fall der Anpassung an den Kontext, definiert Brown kontextsensitive Anwendungen als Anwendungen, die automatisch Informationen bereitstellen und/oder in Aktion treten abhängig vom aktuellen Kontext des Benutzers, der durch Sensoren bestimmt wird[4].

In dieser Arbeit wird eine allgemeinere Version der Definition von Kontextsensitivität verwendet, passend auch zu der Definition von Kontext, beschreiben Dey und Abowd[2]: *„Ein System ist kontextsensitiv, wenn es den Kontext verwendet, um relevante Informationen und/oder Dienstleistungen dem Benutzer anzubieten, wobei die Relevanz von der Tätigkeit des Nutzers abhängt.“*

2 Grundlagen

2.1 Mögliche Ansätze zur Kontext-Modellierung

Um Kontexte innerhalb einer Anwendung verwenden zu können, müssen diese auf eine bestimmte Art und Weise modelliert werden. Zwar ist es ebenfalls möglich, unterschiedliche Kontexte getrennt voneinander zu verarbeiten und zu modellieren, indem zum Beispiel die Uhrzeit getrennt von anderen Kontextdaten in der Anwendung verwendet wird, jedoch bietet es sich an, sobald eine große Anzahl von verschiedenen Kontextdaten betrachtet werden soll, diese in einem einheitlichen (und allgemeinen) Datenmodell zu verwalten. Dadurch kann zusätzlicher Arbeitsaufwand, der durch das Entwerfen und Umsetzen einzelner Kontexte entsteht, vermieden und eine hohe Wiederverwendbarkeit des Codes erreicht werden.

Ein allgemeines Modell muss möglichst eine große Menge an möglichen Kontexten modellieren können, erweiterbar sein, wenn neue Kontexte hinzukommen und natürlich maschinenlesbar und somit für einen Computer verarbeitbar sein. Strang et al.[24] haben die relevantesten Ansätze, diese Anforderungen umzusetzen zusammengefasst als:

Schlüssel-Wert-Paar Modelle
Diese Modelle repräsentieren die einfache Datenstruktur, Kontexte zu modellieren. Sie werden häufig im Kontext von verteilten Service-Frameworks benutzt, wobei die Services selber durch eine Liste simpler Attribute in Form von Schlüssel-Wert-Paaren beschrieben werden. Diese Modelle sind im allgemeinen leicht zu verwalten, sind allerdings nur beschränkt dazu fähig, die Daten gut zu strukturieren.

Markup-Schema Modelle
Markup-Schema Modelle werden durch eine hierarchische Struktur dargestellt, die durch Tags in Zusammenhang mit Attributen und Inhalten aufgebaut wird. Typische Repräsentationen dieser Modelle sind Profile, meist Serialisiert durch eine Ableitung von SGML, die maschinenlesbar Kontextkonzepte beschreiben können. [3][24]

Grafische Modelle
UML und andere grafische Modellierungsmethoden können ebenfalls benutzt werden, um auf grafische Art und Weise Kontext zu modellieren. Bei diesen Ansätzen, wird ausgehend von einem High-Level Modell automatisch die nötige Infrastruktur erzeugt, die notwendig ist, um das Kontextmodell umzusetzen. Diese Modellierung wird hauptsächlich als Grundlage der Modellgetriebenen Entwicklung von kontextsensitiven Systemen verwendet.

Objektorientierte Modelle
Objektorientierte Kontextmodelle sind bemüht, die Vorteile des Programmierparadigmas der Objektorientierung, wie Wiederverwendbarkeit und die Abkapselung der konkreten Daten auch im Bereich der Kontextmodellierung zu nutzen. In diesen Modellen werden standardisierte Interfaces verwendet, um die konkreten Implementierungen der Kontexterkennung in eigene Objekte abzukapseln und einen hohen Grad an Modularität zu erreichen.

Logische Modelle
Logische Modelle bestehen aus Ausdrücken und Fakten, sowie einer Menge von Regeln, die bestimmen, wie weitere Ausdrücke und Fakten aus den vorhandenen abgeleitet werden können. Kontextinformationen werden dem Modell mit Hilfe von CRUD-Funktionen als Fakten hinzugefügt und das Verhalten der Anwendungen wird durch die gegebenen Regeln der formalen Logik abgeleitet und angepasst.

Ontologiebasierte Modelle
Ontologien bieten ein Werkzeug zur Beschreibung von Entitäten, sowie der Relationen zwischen verschiedenen Entitäten, sowie Möglichkeiten, über vorhandenes Wissen neue Schlussfolgerungen in der Ontologie selbst zu ziehen. Da es nach der Definition von Kontext bei der Kontextsensitivität primär um Informationen zu relevanten Entitäten gilt, ist naheliegend, dass Ontologien sich sehr gut zur Modellierung von Kontexten eignen.

Strang et al. evaluierten diese Modellierungsansätze anhand von sechs ermittelten Anforderungen, die insbesondere im Bereich des ubiquitären Rechnens an ein solches Kontextmodell gestellt werden. Diese Anforderungen sind:

Verteilter Aufbau
Da ubiquitäre Systeme in der Regel verteilte Systeme sind und bei diesen im Allgemeinen nicht jeder Part des Systems zu jedem Zeitpunkt Überblick über den Gesamtzustand des Systems hat, sollte ein Kontextmodell verteilbar und synchronisierbar, unabhängig von der konkreten Systemarchitektur und Netzwerktopologie sein.

Partielles Validieren
Das verwendete Kontextmodell sollte es ermöglichen, Regeln und Schemata zu definieren, mit denen die Gültigkeit und Sinnhaftigkeit des gemessenen Kontext zumindest teilweise überprüft werden kann, um zu Verhindern, dass sich Messfehler oder Ähnliches auf die Integrität des Modells auswirken.

Reichhaltigkeit und Qualität der Informationen
Da die Qualität und Aussagekraft verschiedener Sensoren sich über die Zeit ändern kann, sollte ein gutes Kontextmodell die Möglichkeit bieten, die Relevanz und Qualität der gemessenen Werte zu indizieren und somit zu verhindern, dass bekannt ungenaue Werte ebenso stark in die Kontextverarbeitung einbezogen werden, wie genaue.

Unvollständigkeit und Unsicherheit
Da nicht jeder Part eines ubiquitären Systems jederzeit einen Überblick über den Gesamtzustand hat und haben soll, sollte das verwendete Kontextmodell die Möglichkeit bieten auch ohne vollständiges Wissen zum Beispiel mit Hilfe von Interpolation sinnvolle Ergebnisse zu liefern.

Grad der Formalität
Das Kontextmodell sollte unmissverständlich formuliert sein und daher ein Mindestmaß an Formalität aufweisen. Alle Teilsysteme und Anwendungen, die das Modell benutzen, sollten die Informationen auf dieselbe Art interpretieren, um Konsistenz und Korrektheit zu wahren.

Anwendbarkeit auf bestehende Umgebungen
In der praktischen Anwendung sollte man ein Kontextmodell auch auf bereits vorhandene Strukturen, wie Service-Frameworks und Webservices anwenden können, um einfaches Umsteigen auf das neue Kontextmodell zu ermöglichen.

Das Ergebnis der Evaluation nach diesen Kriterien war, dass ontologiebasierte Ansätze dicht gefolgt von der objektorientierten Modellierung die erfolgversprechendsten Aussichten bieten, um Kontext ausdrucksstark zu modellieren und in der Praxis der Software-Entwicklung als Kontextmodellierung zu verwenden. Zu beachten ist an dieser Stelle allerdings, dass die Kriterien, die Strang et al. bei dieser Evaluation angewendet haben, andere für den Entwickler relevanten Größen, wie die Einfachheit oder der Ressourcenaufwand der Kontextmodellierung außer acht lassen und so nicht für alle Projekte zu einer guten Empfehlung führen.

2.2 Die Architektur

Kontextsensitive Systeme können auf viele Arten umgesetzt werden. Die Entscheidung dafür, welche mögliche Architektur man für die eigene Anwendung verwendet, hängt von einer großen Menge von Faktoren ab, wie die Situation der Sensoren (lokal/entfernt), die vorhandenen (System-)Ressourcen auf dem Gerät selber oder die Anzahl an möglichen Nutzern[3]. Ein wichtiger Punkt hierbei ist die Frage, wie Kontextinformationen gewonnen werden sollen. Die drei gängigsten Wege, dies zu erreichen sind[3]:

Direktes Ansprechen der Sensoren
Beim direkten Ansprechen der Sensoren, ist die kontextsensitive Anwendung selber dafür verantwortlich, aus den Sensordaten die Kontextinformationen zu extrahieren. Diese Methode wird häufig benutzt, wenn die Sensoren direkt in das Gerät eingebaut sind, auf dem die Anwendung ausgeführt wird. Nachteil dieser Methode ist, dass die Austauschbarkeit sehr gering ist, da die Treiberschnittstellen der Sensoren direkt angesprochen werden und die Anwendung entsprechend für unterschiedliche Geräte aufwendiger anzupassen ist.

Middleware
Beim Einsatz einer Middleware zur Extraktion der Kontextinformationen, wird eine Schichtenarchitektur verwendet. Die Middleware setzt oberhalb der Hardwareebene auf und trennt damit die eigentliche Geschäftslogik von der genauen Implementierung der Kontexterkennung. Großer Vorteil dieser Methode ist die höhere Wiederverwendbarkeit des Codes der Geschäftslogik sowie die bessere Erweiterbarkeit des Programms um weitere mögliche Kontexte.

Kontext-Server
Eine Erweiterung der bei der Middleware-Architektur umgesetzten Idee, die Akquirierung der Kontextinformationen von der Geschäftslogik zu trennen bildet die Architektur der Kontextserver. Der Unterschied zum klassischen Middlewaregedanken ist es, dass hier mehreren Klienten der Zugriff auf dieselben Daten ermöglicht wird, indem die Extraktion der Kontextinformationen an zentraler Stelle auf einem Server stattfindet und sich die Klienten nur noch die errechneten Kontextinformationen geben lassen müssen. Diese Architektur ist insbesondere in den Fällen sinnvoll, in denen ein verteiltes System entworfen werden soll, die Sensoren die den Kontext ermitteln (teilweise) außerhalb des Gerätes selber sind oder wenn der rechenaufwendige Part der Kontextermittlung nicht auf den oft Leistungsschwachen mobilen Geräten stattfinden soll.

Sowohl in einer Middleware-Architektur, als auch bei der Kontext-Server-Variante gibt es die Möglichkeit auf bereits Implementierte Lösungen zurückzugreifen und auf eine existierende Implementierung aufbauend die Geschäftslogik zu entwickeln. Zu bedenken ist an dieser Stelle allerdings, dass beiden Architekturen bereits intern ein bestimmtes Kontextmodell zugrunde liegt und die Kontexte in einer festgelegten Form von der Middleware/dem Server zurückgegeben werden. Die Wahl einer solchen Implementierung sollte daher auch nach dem Gesichtspunkt erfolgen, welche Form von Kontextmodellierung man selber in der Geschäftslogik verwenden möchte und ob die gebotenen Schnitstellen diese unterstützen.

3 Hauptteil

3.1 Entwicklung mit einem ontologiebasierten Ansatz

Der Begriff der Ontologie hat in der Philosophie eine lange Geschichte, wo er das Thema des Seins und der Existenz beschreibt. In der Informatik auf der anderen Seite, beschreibt das Konzept der Ontologien einen Ansatz, Informationen wie Menschen sie in ihrem Alltag benutzen für den Computer verständlich zu machen. Das Besondere am Konzept der Ontologien ist es, dass zusätzlich zu den konkreten Kontextdaten auch Informationen zu der semantischen Bedeutung dieser Daten in einer Ontologie enthalten sein kann. Diese semantischen Informationen wiederum erlauben es, logische Schlüsse durch automatische Inferenz aus den vorhandenen Daten zu ziehen und so von gegebenen auf neue Fakten schließen zu können.

Formal wird der Begriff Ontologie meist definiert als eine explizite formale Spezifikation einer Konzeptionalisierung [18]. Elemente einer Ontologie sind:

Klassen, die ähnlich wie in der Objektorientierung Ober- und Unterklassen (inklusive Mehrfachvererbung) haben können und gemeinsame Eigenschaften von Konzepten beschreiben.

Typen, die einer Klasse zugeordnet werden können.

Instanzen von Klassen, die das eigentliche Objekt, statt dem Konzept des Objektes beschreiben.

Relationen, durch die Beschrieben wird, in welcher Verbindung einzelne Objekte miteinander stehen.

Axiome mit deren Hilfe Wissen repräsentiert werden kann, das nicht aus anderen Daten abgeleitet werden können. Axiome beschreiben Aussagen, die in der entsprechenden Domäne immer wahr sind.

Außerdem lassen sich Ontologien in Domänenontologien und Ontologien höherer Stufe aufteilen. Domänenontologien beschreiben, wie der Name vermuten lässt

eine spezielle Domäne, die einen Ausschnitt der Welt repräsentiert. Sachverhalte, Relationen und Klassen innerhalb dieser Ontologien beziehen sich stets nur auf die Domäne, die sie beschreiben. Ontologien höherer Stufe hingegen beschreiben Objekte auf eine Art, die in allen Domänen anwendbar ist. In diesen Ontologien wird ebenfalls festgehalten, wie sich Instanzen einer bestimmten Domäne in eine andere Domäne überführen lassen, sodass durch diese Ontologien ein hohes Maß an Kompatibilität zwischen verschiedenen Datenstrukturen und Ontologien erreicht werden kann.

Wie und auf welchem Datenmodell basierend Ontologien umgesetzt werden, ist nicht allgemein spezifiziert. Es gibt eine Vielzahl von Ontologiesprachen, die benutzt werden können, um Ontologien umzusetzen, mit dem Ressource Description Framework (RDF) und der darauf aufbauenden Web Ontology Language (OWL) als einem der am weitesten verbreiteten Vertreter. Die Popularität von OWL als Beschreibung von Ontologien ist insbesondere auf die Semantic Web-Bewegung des World Wide Web Consortiums (W3C) zurückzuführen, in deren Kontext OWL erstmals definiert wurde, um bestimmte Aspekte des semantischen Internets umzusetzen. Diese Popularität von RDF und OWL lässt sich auch bei den meisten ontologiebasierten Kontext-Management-Ansätzen beobachten, bei denen in der Regel OWL verwendet wird, um dem Begriff Kontext in Form einer Domänenontologie computerlesbare semantische Bedeutung zuzuweisen.

OWL und RDF Die Web Ontology Language ist eine semantische Web-Sprache, die speziell dafür entwickelt wurde, von Anwendungen benutzt zu werden, die den Inhalt der abgefragten Informationen selber verarbeiten und nicht nur einem menschlichen Benutzer anzeigen sollen, was sie von anderen Web-Sprachen, wie zum Beispiel HTML abgrenzt. Technisch basiert OWL auf der RDF-Syntax, erweitert diese jedoch um weitere Sprachkonstrukte, die es erlauben, Ausdrücke ähnlich der Prädikatenlogik zu formulieren und darauf basierend Inferenz zu ermöglichen.

RDF selber wiederum beschreibt ein Datenmodell in dem Ontologien gespeichert werden können. Im RDF-Modell werden alle Informationen in Form eines Graphen gespeichert. Jede Kante in dem Graphen repräsentiert hierbei eine Aussage der Form Subjekt - Prädikat - Objekt, die sich in Tripelschreibweise festhalten lässt. Sowohl Subjekt als auch Prädikat sind selber wiederum über eine URI eindeutig identifizierbare Ressourcen im Datenmodell und können in anderen Tripeln an selber oder anderer Stelle auftauchen. Objekte hingegen können entweder eine Ressource oder ein sogenanntes Literal sein, eine einfache Zeichenkette mit Optionaler Angabe dazu, was für einen Datentyp sie repräsentiert. Abgelegt werden RDF-Daten in Triplestores, speziellen Datenbanken, die auf die Speicherung solcher Graphenstrukturen spezialisiert sind und Möglichkeiten zur Abfrage in passenden Sprachen, wie die SPARQL Protocol and RDF Query Language (SPARQL) zur Verfügung stellen. [10]

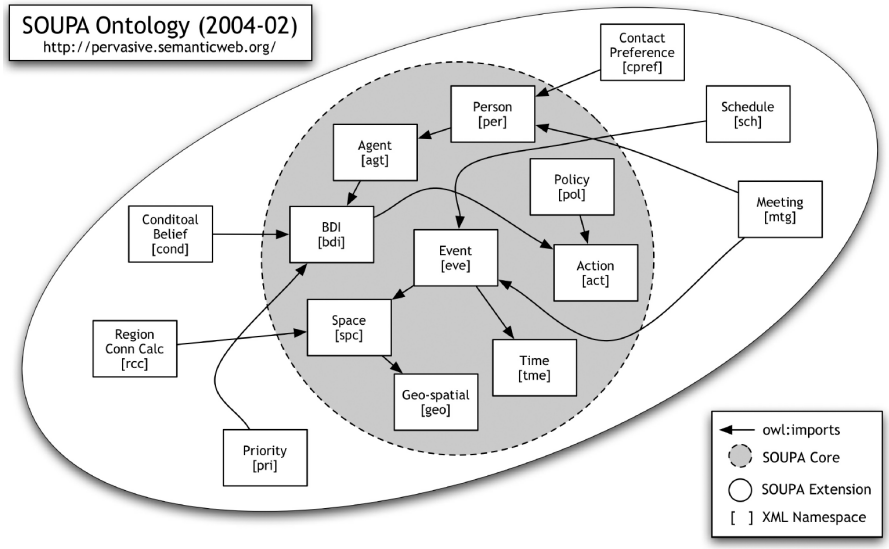


Abbildung 1. SOUPA ist zusammengesetzt aus einem Kern (Core), sowie Erweiterungen des Kerns (Extensions). Unterschiedliche Domänenspezifikationen werden durch unterschiedliche XML-Namensräume voneinander getrennt[10].

CoBrA und SOUPA Ein Beispiel für die Umsetzung des Kontextmodells mit Hilfe von Ontologien und OWL bildet die Context Broker Architecture (CoBrA), basierend auf der Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA). Das Ziel des SOUPA-Projektes ist es, eine möglichst allgemeingültige Ontologie zu definieren, die besonders gut für die Modellierung von Kontexten geeignet ist und zudem eine schnelle automatische Inferenz ermöglicht. Die Ontologien, die durch SOUPA beschrieben werden, lassen sich, wie in Abbildung 1 dargestellt, aufteilen in den SOUPA-Kern (Core), welcher allgemeine und universelle Konzepte, die zum Entwickeln von kontextsensitiven und ubiquitären Anwendungen nötig sind, beinhaltet, sowie die SOUPA-Erweiterungen (Extensions), die auf den Kern-Ontologien aufbauen und dafür gedacht sind, spezifischere Anwendungstypen zu unterstützen und als Beispiel für die Definition weiterer Ontologien zu dienen. Um die unterschiedlichen Anwendungstypen intern zu trennen und Verwechslungen zu vermeiden, werden zusätzlich unterschiedliche, von RDF unterstützte XML-Namensräume verwendet.

Bei der Modellierung der einzelnen Konzepte im SOUPA-Kern wurde sich außerdem an vielen Stellen an andere weit verbreiteten Ontologien orientiert, ohne diese direkt zu importieren, obwohl das Importieren von Konzepten aus anderen Ontologien ohne größeren Aufwand möglich ist. Dies hat den Hintergrund, dass das Importieren dieser Konzepte aus anderen Ontologien auch zum Import vieler für die Kontextdomäne irrelevanten Daten, Eigenschaften und Relationen, die in

den anderen Ontologien direkt mit diesen Konzepten zusammenhängen, führen würde, wodurch ein zusätlicher Overhead entsteht, der das automatische Schlussfolgern von Fakten aus der Ontologie erheblich verlangsamen kann. Stattdessen wurde bei den entsprechenden Klassen eine Anmerkung gemacht, zu welchen externen Konzepten die entsprechende Klasse äquivalent ist, um dennoch eine hohe Portabilität zu erhalten und anderen Ontologien die Möglichkeit zu bieten, die SOUPA-Konzepte zu importieren.[10]

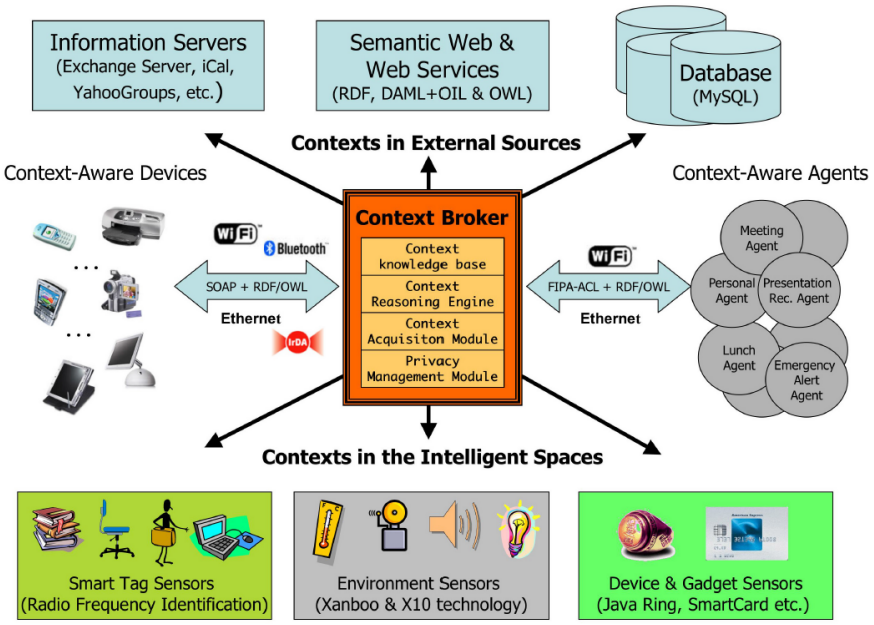


Abbildung 2. Darstellung der zentralisierten Context Broker Architecture.[8].

Diese SOUPA-Ontologie bildet das Grundlegende Datenmodell in der Context Broker Architecture, eine agentenbasierte Architektur nach dem Kontext-Server-Modell, die Kontextsensitivität in intelligenten Umgebungen unterstützt. Intelligente Umgebungen bezeichnen in diesem Fall physische Umgebungen, die mit (intelligenten) Systemen aus dem Bereich des Pervasive Computings ausgestattet sind. Die wichtigste Rolle in dieser in Abbildung 2 dargestellten Architektur spielt der sogenannte Kontext-Vermittler (Broker), der auf einem stationären, ressourcenreichen Server läuft und die allgemeine Verwaltung des Datenmodells und der Kontextverwaltung für die Agenten übernimmt.[10] Die Agenten auf der anderen Seite sind leichtgewichtige Anwendungen die auf mobilen Geräten, die vom Benutzer getragen oder benutzt werden (z.B. Smart Phones, PDAs etc.), Dienste, die im Rahmen der intelligenten Umgebung arbeiten (z.B. Lichtkon-

trolle, Temperaturkontrolle etc.), oder Webservervices, die eine Netzpräsenz für Menschen, Orte oder Objekte in der Umgebung anbieten (z.B. Dienste, die die Bewegung und den Ort von Objekten und Personen verfolgen)[7].

Die Hauptaufgaben des Kontext-Vermittlers sind es, (i) ein zentralisiertes Kontextmodell das auf allen Geräten und Services in der Umgebung geteilt werden kann, zur Verfügung zu stellen, (ii) Kontextinformationen von Quellen, die von den Agenten nicht direkt angesprochen werden können, zu sammeln, (iii) Schlussfolgerungen die nicht direkt von den Sensoren abgelesen werden können, zu treffen, (iv) Inkonsistenzen im geteilten Wissen der Agenten zu erkennen und zu beheben und (v) die Privatssphäre der Benutzer zu schützen, indem dem Benutzer überlassen wird, mit welchen Agenten die eigenen Kontextinformationen geteilt werden. Umgesetzt werden diese Anforderungen verteilt auf vier miteinander zusammenarbeitenden Komponenten[7]:

Context Knowledge Base
Ein persistenter Speicher des Kontextwissens, der eine Menge von APIs anbietet über die andere Komponenten auf das gespeicherte Wissen zugreifen können. Außerdem sind hier die Informationen und Ontologien der entsprechenden intelligenten Umgebung gespeichert, sowie heuristisches Wissen, das benötigt wird, um Schlussfolgerungen ziehen zu können und die Daten konsistent zu halten.

Context Reasoning Engine
Ein Modul, welches in der Lage ist, auf Grundlage von Ontologien deduktiv Schlussfolgerungen ziehen zu können. Mit Hilfe der heuristischen Informationen ist es zudem möglich, Inkonsistenzen im Datenbestand zu erkennen und zu beheben.

Context Acquisition Module
Dieses Modul bildet eine Schicht zwischen dem Rest des Kontext-Brokers und den hardwarenahen Sensorimplementierungen. Hier wird festgehalten, auf welche Weise Kontextdaten aus konkreten Sensordaten gewonnen werden.

Policy Management Module
Die Aufgabe dieses Moduls ist es, die Anforderungen an Sicherheit und Privatheit umzusetzen. In diesem Modul werden Inferenzregeln festgehalten mit denen die Benutzerverwaltung Rechte vergibt. Diese Regeln entscheiden außerdem, welche Recheneinheiten welche Rechte haben und welche Agenten bei bestimmten Kontextänderungen informiert werden sollen.

Um CoBrA mit allen angestrebten Zielen der Architektur umzusetzen und in einem eigenen System zu verwenden, ist somit die Implementierung dieser vier Module, sowie die Implementierung der einzelnen Agenten, die in der entsprechenden intelligenten Umgebung agieren sollen, notwendig. Eine Beispielimplementierung unter der Creative Commons Lizenz gibt es bereits von Seiten der

Autoren der CoBrA. Die Context-Reasoning Engine basiert in dieser Implementierung auf einer Inference-Engine namens F-OWL, welche aus OWL basierenden Datenbeständen Schlussfolgerungen ziehen kann. [9]

3.2 Logik-basierter Ansatz

Logikbasierte Ansätze zur Verarbeitung von Kontextdaten weisen einige Gemeinsamkeiten mit den ontologiebasierten Ansätzen auf, da auch auf logischen Datenbeständen automatische Inferenz möglich ist und beiden ein hohes Maß an Formalität innewohnt. Anders als bei den Ontologien, werden logische Datenbestände allerdings nicht aus semantischen Beschreibungen der Umwelt aufgebaut, sondern aus logischen Aussagen und Fakten, aus denen nach den Grundlagen des Programmierparadigmas der Logischen Programmierung Schlussfolgerungen über den Kontext gezogen werden.

Logische Grundlagen Üblicherweise beschränkt man sich bei den so benutzten logischen Aussagen auf eine vereinfachte Form der Prädikatenlogik erster Stufe, wie die Reduzierung auf Aussagen in konjunktiver Normalform (KNF) oder als eine Menge von Hornformeln. Mit Hilfe der Prädikatenlogik ist es möglich, die logische Struktur von Prämissen zu beschreiben, indem diese analysiert und in Objekte und Prädikate, die sich auf diese Objekte beziehen, umgewandelt werden. Prämissen beschreiben in diesem Kontext elementare Fakten, während Prädikate atomare Beschreibungen der Eigenschaften von Gegenständen - Grundbestandteilen der „Welt“ mit bestimmten Eigenschaften - repräsentieren. Außerdem sind in der Prädikatenlogik allgemeine Aussagen über die logischen Zusammenhänge der Prädikate in Form von Allquantoren und Existenzquantoren möglich.

Die Prädikatenlogik erlaubt es somit verschiedene Formen von Aussagen zu treffen, wie zum Beispiel[14]:

Fakten
Einfache Fakten, so wie „Daniel befindet sich in Raum -101“. Diese Aussage bezieht sich sowohl auf das Objekt „Daniel“, als auch auf das Objekt „Raum -101“ und kann formal notiert werden als:

$$befindetSichIn(Daniel, Raum - 101)$$

allgemeine Aussagen
Eine Aussage über alle Objekte eines bestimmten Types oder mit einer bestimmten Eigenschaft, wie „Alle Menschen sind Benutzer“. Solche Aussagen lassen sich formal mit dem Allquantor notieren:

$$\forall x : istMensch(x) \Rightarrow istBenutzer(x)$$

vage Aussagen
Die Feststellung, dass es ein Objekt mit bestimmten Eigenschaften gibt, ohne zu spezifizieren, um welches Objekt es sich handelt, wie „Es gibt einen Ort an dem sich Daniel aufhält." formalisierbar durch den Existenzquantor:

$$\exists x : befindetSichIn(Daniel, x) \wedge istOrt(x)$$

komplexe Aussagen
Aussagen, die das Verhältnis zwischen mehreren Objekten mit bestimmten Eigenschaften beschreiben, wie „Wenn sich ein Objekt in einem Raum aufhält, der sich in einem Gebäude befindet, hält sich die Person ebenfalls in dem Gebäude auf." Solche generellen Aussagen erlauben logische Schlüsse, wie zum Beispiel „Daniel hält sich in Raum -101 auf, Raum -101 befindet sich in Gebäude 50.34, daraus folgt Daniel befindet sich in Gebäude 50.34." Formal zu notieren als:

$$\forall x, y, z : befindetSichIn(x, y) \wedge befindetSichIn(y, z) \Rightarrow befindetSichIn(x, z)$$

Sind ausschließlich allgemeine Aussagen über Objekte und Individuen und nicht ebenso über Prädikate wie $istMensch(x)$ oder $befindetSichIn(x, y)$ möglich, so spricht man von einer Prädikatenlogik erster Stufe. [14]

Logisches Programmieren Beim Programmierparadigma der logischen Programmierung benutzt man Prädikatenlogik erster Stufe zusammen mit automatischer logischer Inferenz als Mittel zur Problemlösung. Beim logischen Programmieren beschreibt man einen (logischen) Sachverhalt durch eine Menge von Aussagen und überprüft, ob durch Inferenz Schlussfolgerungen möglich sind, die zur Lösung des gestellten Problems beitragen. Ein großes Problem dieses Ansatzes besteht allerdings durch die hohe Komplexität der Inferenz. Anhand von gegebenen logischen Aussagen ist es nicht möglich, automatisch zu erkennen, welche Inferenzen über die Aussagen zu einer Lösung des Problems führen, oder welche Inferenzen dies am schnellsten erreichen können. Daher ist es notwendig, Strategien und Meta-Regeln, in welcher Reihenfolge die Inferenzen überprüft werden sollen, in die Inferenz-Engine, die das automatische Schlussfolgern übernimmt, einzubauen und für den entsprechenden Problembereich auszuwählen. Des weiteren versucht man beim logischen Programmieren meistens, den vollen Umfang der Syntax der Prädikatenlogik (erster Stufe) zu vermeiden, indem man die erlaubten Eingaben auf Aussagen in bestimmten Formaten, wie die konjunktive Normalform (insbesondere Horn-Formeln) beschränkt und somit die Anzahl der benötigten Strategien, die bestimmen, wie die gegebenen Aussagen bei der Inferenz umgewandelt werden, reduziert und damit auch die Komplexität der Inferenz-Engine verringert. Im Falle einer Reduktion der Aussagen auf Horn-Formeln ist es sogar möglich mit nur einer einzigen Strategie, dem so genannten Resolutions-Verfahren, auszukommen, um (mit quadratischem Zeitaufwand und falls möglich) eine Lösung für das gegebene Problem zu finden. Da die meisten

Ansätze zur logischen Programmierung, einschließlich der am weitesten verbreiteten Programmiersprache dieses Paradigmas - Prolog, Horn-Formeln zum Finden der Lösung eines logischen Problems verwenden, werden diese im Folgenden genauer betrachtet. Horn Formeln bestehen aus einer Menge von Horn-Klauseln, die miteinander konjunktiv verbunden sind. Eine Horn-Klausel wiederum besteht aus einer Disjunktion von Literalen ϕ_i, von denen jeweils der atomare Ausdruck α_i des Literals bei maximal einem Literal nicht negiert ist:

$$\text{Horn-Formel: } \phi_1 \wedge ... \wedge \phi_n$$
$$\text{Horn-Klausel: } \neg\alpha_1 \vee ... \vee \neg\alpha_{n-1} \vee \alpha_n$$

Die Horn-Klauseln sind hierbei so aufgebaut, dass sie von der Bedeutung her einer Implikation entsprechen und damit grob als logisches Äquivalent von if-Bedingung aus anderen Programmiersprachen gleichzusetzen sind:

$$\text{Horn-Klausel: } (\alpha_1 \wedge ... \wedge \alpha_{n-1}) \Rightarrow \alpha_n$$

Diese Form lässt bereits vermuten, dass sich Horn-Klauseln gut dazu verwenden lassen, Bedingungen zu formulieren, in denen ein bestimmter Kontext eintrifft. Die Menge aller Kontexte mitsamt ihrer Eintrittsbedingungen lässt sich somit in einer Hornformel zusammenfassen, die genau dann wahr ist, wenn es einen Kontext gibt, für den alle Bedingungen erfüllt sind.

Regelbasierter Ansatz Ein Speziallfall des logischen Ansatzes ist der Regelbasierte Ansatz. Hier reduziert man die Prädikatenlogik auf eine ähnliche Form, wie sie durch die Hornformeln gegeben ist - einer Menge von Bedingungen und dazugehörigen Schlussfolgerungen. Bei diesen Ansätzen wird die Wissensbasis aufgeteilt in eine Menge von Regeln, sowie Aktivierungsbedingungen, die bestimmen, wann diese Regeln angewendet werden sollen. Eine mögliche Umsetzung dieses Ansatzes bietet das Event-Control-Action (ECA) Makro-Entwurfsmuster. Wie in Abbildung 3 dargestellt, trennt es zwischen der Aufgabe der Kontextverarbeitung (Event-Modul) und der Logik, Aktionen als Antwort auf eine Kontextänderung auszulösen (Action-Modul). Diese getrennten Aufgaben werden unter der Kontrolle einer Anwendungsverhaltensbeschreibung (Control-Modul) realisiert, in dem die Verhaltensweisen der kontextsensitiven Anwendung im Form von ECA-Regeln beschrieben werden, die grob die Form haben: if<Bedingung>$then$<Aktion> und somit Hornklauseln entsprechen, bei denen Bedingung und Aktion erfüllt sein müssen, um zu einer wahren Aussage zu führen. Dabei ist der Bedingungspart der ECA-Regel aus einer logischen Komposition von Events zusammengesetzt und spezifiziert die Situation in der die entsprechende Aktion ausgelöst werden soll. Die Events hingegen modellieren ein Ereignis von Interesse in der Anwendung oder Umgebung, während der Aktionsteil der Regel sich aus einer oder mehreren Aktionen zusammen, die ausgelöst werden sollen, wenn der Bedingungspart erfüllt ist.[12][11]
Diese Trennung der Aufgaben wurde in diesem Entwurfsmuster eingeführt, um Probleme der Kontextverwaltung, wie das Erkennen und Verarbeiten des Kontextes selber von den Aufgaben der Reaktionen auf diese unter der Kontrolle

eines Anwendungsmodells, das die Verhaltensweise der Anwendung mit Hilfe der ECA-Regeln definiert, abzukoppeln. Dies hat den Vorteil, dass die Herausforderungen bei der Implementierung insbesondere der Kontextverarbeitung und den Aktionen, die ausgelöst werden sollen einander nicht beeinflussen und unabhängig voneinander implementiert werden können. Da sowohl die Kontextverarbeitung, als auch die ausgelösten Aktionen potentiell beliebig komplex in Struktur und Umfang sein können und nur auf einer hohen Ebene die Architektur beschreibt, handelt es sich beim ECA-Entwurfsmuster um ein Makro-Entwurfsmuster.

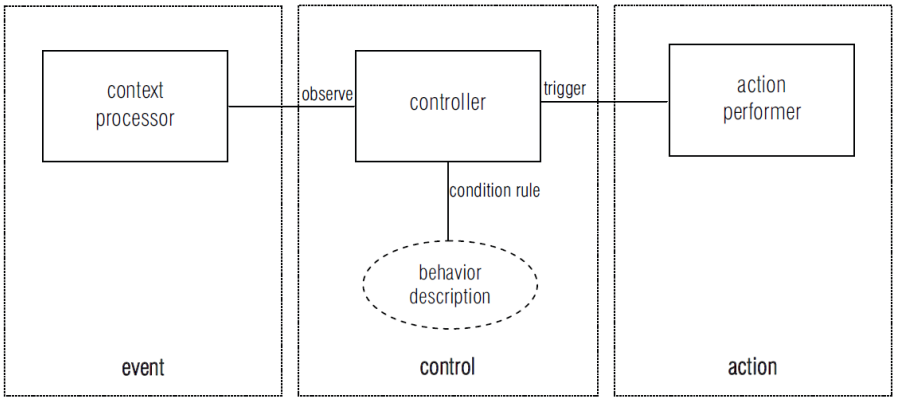


Abbildung 3. Schema dss Event-Control-Action Entwurfsmusters.[16]

Das Event-Modul
Dieses Modul ist dafür verantwortlich, aus Sensordaten Events zu generieren, die jeweils eine relevante Kontextänderung repräsentieren. Für jedes mögliche elementare Event wird ein Kontextmanager entwickelt, der aus konkreten Sensordaten abliest, ob das entsprechende Event eintritt. Komplexere, zusammengesetzte Events können ebenfalls mit Hilfe des ECA-Entwurfsmsuters auf Manager-Ebene durch Rekursion des Entwurfsmusters realisiert werden. Dies geschieht, indem die zusammengesetzten Events durch ECA-Regeln aus anderen Events zusammengesetzt werden und, falls die Regeln erfüllt werden, ein neues, zusammengesetztes Event generieren.

Das Control-Modul
ECA-Regeln werden im Control-Modul verwaltet überprüft und angewendet. Wird eine ECA-Regel durch eine Menge von Events ausgelöst, so wird in diesem Modul die Aktion angestoßen, die als Reaktion auf das Erfüllen der Regel ausgelöst werden soll. Um diese Regeln zu verwalten wird üblicherweise eine Wissensbasis benutzt in der die ECA-Regeln in einer normalisierten

Form abgespeichert werden. Zum überprüfen der Regeln kann eine Rule-Engine, wie JESS[17] oder MANDARAX[15] verwendet werden, die Vor- und Nachteile von diesen, werden ausführlich in [12] besprochen.

Das Action-Modul
Im letzten Modul werden die Aktionen definiert, die ausgelöst werden können. Da solche Aktionen sich in ihrem Wesen und in ihrer Komplexität sich stark unterscheiden können, kann hier nur bedingt eine allgemeine Struktur des Moduls festgelegt werden. Alle Aktionen teilen sich eine gemeinsame, dem Control-Modul bekannte, Schnittstelle, über die spezifiziert wird, welche Aktion mit welchen Parametern ausgelöst werden soll. Optional ist es auch hier möglich, koordinierte Aktionen zu definieren, die sich aus einer Reihe anderer Aktionen zusammensetzen, jedoch muss jede elementare Aktion ähnlich wie die elementaren Events einzeln definiert und programmiert werden.

3.3 Modellgetriebene Entwicklung

Das Prinzip der modellgetriebenen Entwicklung von Anwendungen beschreibt ein Vorgehensmodell zur Softwareentwicklung, das sich auf den Aufbau eines Domänenmodells der zu entwickelnden Software konzentriert, statt auf Rechenkonzepte und Algorithmen. Im Vergleich zu vielen anderen Ansätzen, erfolgt bei der modellgetriebenen Vorgehensweise wesentlich größerer Anteil der Entwicklung in einem plattform-unabhängigen Modell, ohne Code- oder Implementierungsdetails zu spezifizieren, um Redundanz insbesondere bei Problembeschreibungen zu verringern und für klare Definitionen von Domänenkonzepten führt.

Die Modellierungssprache, die in den meisten Fällen benutzt wird, ist die weit verbreitete Unified Modeling Language (UML). Die Benutzung von UML hat den Vorteil, dass es UML durch Profile und Metamodelle ermöglicht, Modellrepräsentationen von Kontexten in unterschiedlichen Domänen anwenden zu können und, dass es durch die weite Verbreitung in der Softwareentwicklung mehrere und gut dokumentierte Codegeneratoren gibt, die aus UML-Modellen ausführbaren Code erstellen können und somit nur wenig Arbeit am eigentlichen Code notwendig ist.

ContextUML Einer der ersten Ansätze die Interaktion zwischen Kontexten und Diensten zu modellieren, wird durch ContextUML beschrieben[23]. Mit ContextUML wird ein UML-Metamodell aufgebaut, das die existierende UML-Syntax durch die Erweiterung mit passenden Artefakten, wie in einer durch Prezerakos et al. erweiterten Form in Abbildung 4 gezeigt, an die Domäne der kontextsensitiven Dienste und Anwendungen anpasst. Von ContextUML abgeleitete Modelle sind Klassendiagramme, bei denen die Klassen sowohl Kontexte, als auch Anwendungskonstrukte darstellen, während Assoziationen und Abhängigkeiten zwischen den einzelnen Klassen beschreiben, wie die Interaktion zwischen

den Anwendungen und den Kontexten aussieht. Zusätzlich zu den Klassendiagrammen können bei ContextUML UML-Aktivitätsdiagramme verwendet werden, um die Kernlogik der Dienste zu modellieren.[20]

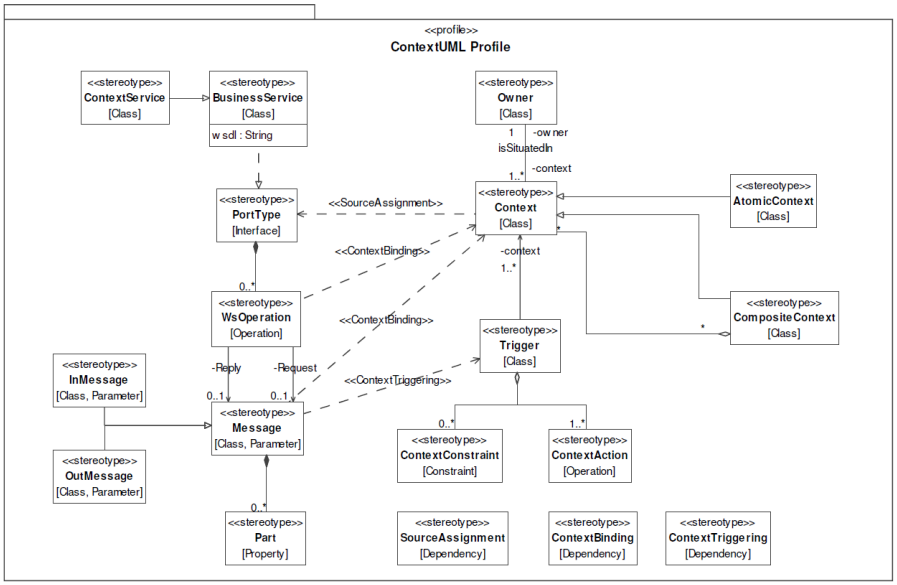


Abbildung 4. Klassendiagramm des erweiterten ContextUML-Metamodells nach Prezerakos et al. [21] mit loser Kopplung zwischen den Dienskomponenten beschreibenden Klassen und den Kontext-Klassen.

Kernelement des ContextUML-Metamodells ist naheliegenderweise der Stereotyp Context. Objekte dieser Klasse modellieren die Kontextinformationen zu einem Owner-Objekt, die innerhalb der Anwendung verwendet werden sollen um zu entscheiden, wie gehandelt wird. Um auch komplexere Kontextgebilde zu ermöglichen, lässt sich innerhalb des Typs Context differenzieren zwischen nicht weiter aufteilbaren AtomicContext-Objekten und CompositeContext-Objekten, die in Form einer Baumstruktur aus anderen Kontexten aufgebaut sind und so eine ganze Menge von Kontextinformationen repräsentieren, unterscheiden.

Um automatische Handlungen der Anwendung anzustoßen gibt es den Stereotypen der Trigger-Klasse. Ein Trigger wird jeweils benutzt, um für den gegebenen Kontext zu überprüfen, was ausgeführt werden soll und setzt sich zusammen aus einer Menge von ContextConstraints, die spezifizieren, unter welchen Bedingungen der Trigger ausgelöst wird, sowie mindestens einer ContextAction durch die bestimmt wird, welche Handlungen bei Erfüllung aller Contraints ausgeführt werden sollen.

Des Weiteren werden insbesondere durch die von Prezerakos et al. erweiterten Form von ContextUML Grundlegende Stereotypen der Geschäftslogik (linker Teil von Abbildung 4) definiert, die verwendet werden können und sollen, um die Context- und Trigger-Klassen in die Geschäftslogik der Anwendung einzubauen. Die Typen WSOperation, sowie Message mit all ihren Untertypen sind die eigentlichen kontextsensitiven Objekten, denen durch die ContextBinding-Abhängigkeit jeweils ihr aktueller Kontext zugewiesen wird.

4 Diskussion

Die hier vorgestellten Herangehensweisen, Kontext und Kontextsensitivität zu modellieren und umzusetzen unterscheiden sich in ihren Herangehensweisen und den jeweiligen Umsetzungen zum Teil stark voneinander. In diesem Kapitel soll untersucht werden, für welche Arten von kontextsensitiven Anwendungen sich ein Entwickler für die einzelnen Herangehensweisen entscheiden sollte. Insbesondere soll hierbei der Fokus nicht auf den konkreten Eigenschaften bestimmter Umsetzungen der besprochenen Modellierungsansätze, wie zum Beispiel die Flexibilität der Kontextmodellierung in der SOUPA-Ontologie, liegen, da die konkreten Umsetzungen hier nur Beispielhaft angeführt werden und sich teilweise andere Realisierungen mit anderen Stärken und Schwächen finden lassen, sondern auf für den Entwickler relevanten Eigenschaften, die den einzelnen Modellierungen an sich innewohnen.

4.1 Die Kriterien des Vergleichs

Die Kriterien, die hier verwendet werden sind deswegen so gewählt, dass ein Entwickler bereits zu Beginn der Entwicklung der Anwendung wissen kann, inwiefern sie für die zu entwickelnde Anwendung relevant sind:

Ressourcensparung
Viele kontextsensitive Anwendungen sollen auf mobilen Geräten mit stark beschränkten Ressourcen laufen und haben deswegen nur wenig Prozessorleistung, Arbeitsspeicher und oft keine Möglichkeiten zur parallelen Ausführung zur Verfügung. Soll ein kontextsensitives System daher ausschließlich auf einem solchen Gerät laufen, sollte ein Ansatz ausgewählt werden, der möglichst wenig Ressourcen auf dem Gerät erfordert. Stehen auf der anderen Seite ausreichend Ressourcen auf dem Zielgerät zur Verfügung, oder ist eine Client-Server-Architektur denkbar, kann dieser Punkt eher vernachlässigt werden.

Wiederverwendbarkeit
In vielen Domänen, in denen kontextsensitive Systeme zum Einsatz kommen, wie zum Beispiel im Smarthome-Bereich, besteht Interesse daran, mehrere Anwendungen in Kooperation miteinander zu verwenden, oder zu einem

späteren Zeitpunkt weitere kontextsensitive Anwendung in dieser Domäne zu schreiben. Ist dies der Fall, so bietet es sich an, das vorher implementierte Kontextmodell wiederzuverwenden. Handelt es sich hingegen um eine für sich stehende Anwendung in einer abgeschlossenen Domäne, so kann dem niedrige Priorität beigemessen werden.

Erweiterbarkeit
Ein flexibles Kontextmodell bietet die Möglichkeit, zu einem späteren Zeitpunkt ohne großen Aufwand erweitert zu werden. Falls der Benutzer selber zum Anwendungszeitpunkt Kontextreaktionen im Rahmen einer Konfiguration definieren können soll, oder wenn abzusehen ist, dass dem Kontextmodell zu einem späteren Zeitpunkt neue Kontextdefinitionen hinzugefügt werden, so sollte ein Modell gewählt werden, das eine gute Erweiterbarkeit aufweist.

Interoperabilität
Interoperabilität bezeichnet die Fähigkeit mit anderen, verschiedenen Systemen zusammenzuarbeiten. Soll die zu entwickelnde Anwendung zusammen mit fremden Anwendungen benutzt werden können, sollte das verwendete Kontextmodell ebenfalls klar genug definierte Schnittstellen aufweisen, um Kontextdaten von anderen Anwendungen in das Modell überführen zu können.

Einfachheit
Je nach Umfang des Projektes, kann für den Entwickler ebenfalls relevant sein, wie einfach die einzelnen Modelle zu implementieren sind und wie viele Vorkenntnisse benötigt werden, um das entsprechende Modell sinnvoll in die Anwendung zu integrieren. Insbesondere bei kleineren Projekten, die schnell zu einem Ergebnis führen sollen, sollte daher eher ein Modell verwendet werden, das eine geringe Einarbeitungszeit benötigt und ohne viel Aufwand implementiert werden kann.

Automatische Schlussfolgerungen
Es ist bei einigen Modellen möglich, sich einen Teil der Kontextermittlung und Analyse durch automatische Inferenz abnehmen zu lassen. Dies ist insbesondere in den Fällen hilfreich, in denen man relativ leicht feste Regeln aufstellen kann, wie die Kontextinformationen verarbeitet werden sollen, da das Modellieren dieser Regeln generell weniger aufwendig ist, als eine vollständige Implementierung dieser Schlussfolgerungen für alle Instanzen in denen sie angewendet werden muss.

4.2 Vergleich der Ansätze

Ressourcenaufwand Um die Vorteile der Modellierung auf Basis von Ontologien nutzen zu können, ist es notwendig die Domäne der Anwendung zu modellieren und teile dieser Modellierung im Arbeitsspeicher zu halten. Des Weiteren wird eine Inferenz-Engine benötigt, die auf dem Ontologiedatenbestand das automatische Reasoning übernimmt. Auch wenn es mittlerweile Ansätze gibt, leichtgewichtige RDF-Speicher für mobile Geräte, wie RDFOnTheGO [1] zu entwickeln, sind diese zu diesem Zeitpunkt noch sehr Speicheraufwendig und ungeeignet für einen Großteil der Mobilgeräte. Zudem benötigen Inferenz-Engines zu diesem Zeitpunkt viel Rechenleistung, um in annehmbarer Zeit Ergebnisse zu liefern, weshalb der Ontologieansatz als sehr Ressourcenaufwendig einzuschätzen und eher für verteilte Client-Server-Architekturen geeignet ist.

Logische Ansätze benötigen ebenso einen Speicher in dem die Fakten und Regeln abgelegt sind, haben dabei allerdings wesentlich weniger Overhead im Vergleich zu Ontologiemodellen, da alle semantischen Informationen, die über das Schließen von Kontextänderungen hinausgehen, nicht abgespeichert sind. Die Inferenz-Engine ist ebenfalls weniger aufwändig als bei den Ontologien, da sich in den meisten logischen Ansätzen die Inferenzregeln auf die Resolutionsregel gemeinsam mit Tiefensuche und Backtracking (im Falle von Prolog) reduzieren lassen.

Der Modellgetriebene Ansatz benötigt keine weiteren Bibliotheken oder Datenbanken, um Kontext verarbeiten zu können. Objekte beziehungsweise Code, die den Kontext repräsentieren sind explizit im generierten Quellcode enthalten und benötigen entsprechend keinen zusätzlichen Ressourcenaufwand.

Wiederverwendbarkeit Ontologien stellen eine semantische Beschreibung der Umwelt dar. Diese Beschreibung wird zwar bei der konkreten Umsetzung einer Anwendung mit bestimmter Kontextdomäne mit Eigenschaften und Informationen versehen, die für andere Programme eventuell irrelevant sind, dennoch wird das Kontextmodell an sich dadurch nicht falsch. Möchte man zu einem späteren Zeitpunkt eine Anwendung auf diese Ontologie aufbauen, so kann diese entsprechend ohne großen Aufwand direkt übernommen und erweitert werden. Ist die Ontologie entsprechend gut konzipiert, kann es sogar möglich sein, dieselbe Wissensbasis, die bereits in Benutzung für eine alte Anwendung ist, ohne Änderungen und ohne Aufwand weiterzuverwenden.

Logische Systeme auf der anderen Seite haben die semantischen Informationen nicht explizit wie bei Ontologien gespeichert, sondern lediglich eine Menge von Regeln und Fakten. Entsprechend kann es sich als schwierig oder zumindest sehr kompliziert herausstellen, die logische Wissensbasis für neue Zwecke anzupassen und sauber zu trennen, welche Regeln weiterhin benutzt werden können, welche abgeändert werden müssen und welche mittlerweile überflüssig geworden sind und entsprechend nur die Effizienz der automatischen Inferenz verringern. Benutzt man die vereinfachte Form der Regelbasierten Systeme, kann die Wiederverwendbarkeit höher sein, da sich die Regeln hier relativ einfach bestimmten

Kontexten und Kontextänderungen zuordnen lassen und auch die Implementierung des Event-Moduls für spätere Anwendungen relevant sein können.

Die Modellgetriebene Entwicklung benutzt für die Modellierung von Kontexten Kontextmetamodelle, die bereits darauf ausgelegt sind, für unterschiedliche Anwendungen benutzt werden zu können. Will man zu einem späteren Zeitpunkt eine neue kontextsensitive Anwendung in ähnlicher Domäne entwickeln, können die Modelle der alten Anwendung beliebig und unkompliziert angepasst oder erweitert werden. Wie gut die effektive Wiederverwendbarkeit bei diesem Ansatz letzten Endes tatsächlich ist, kann hier allerdings stark von der konkreten Umsetzung abhängen, da durch die Metamodelle nur Teile der Architekturen definiert sind und eine ungünstige Modellierung im plattformunabhängigen Modell schwer wiederzuverwenden oder anzupassen ist.

Flexibilität Beim ontologiebasierten Ansatz gibt es einen Ontologiespeicher in dem das semantische Wissen über die Umwelt festgehalten wird und in den meisten Implementierungen wie andere Datenbankformen einfach durch entsprechende Interfaces um Klassen, Instanzen und Relationen erweitert werden kann. Eine Erweiterung der Wissensbasis bei CoBrA würde beispielsweise keinerlei Änderungen für die vorhandenen Agenten bedeuten und eine Änderung von Relationen und Eigenschaften von Instanzen wird von allen Agenten in der Zukunft berücksichtigt. Dies macht den ontologiebasierten Ansatz sehr flexibel im Hinblick auf nachträgliche Änderungen an Wissen und Kontextmodell.

Auch beim logischen Modell wird eine Wissensbasis verwendet, die ohne Probleme um weitere Regeln und Fakten erweitert werden kann. Das Verhalten der Anwendung kann durch eine Änderung dieser Wissensbasis angepasst und erweitert werden, ohne den Code der Anwendung selber anzurühren, was diese Modellierung als sehr flexibel auszeichnet.

Das Ergebnis des modellgetriebenen Ansatzes ist ausführbarer Quellcode für die einzelnen Kontext-Instanzen. Erweiterungen dieses Modells ohne Änderungen der Anwendung sind daher nicht inhärent möglich, sondern nur durch Code-Nachladen oder ähnliche Umwege zu erreichen. Daher ist dieser Ansatz als ziemlich unflexibel zu bewerten.

Interoperabilität Das Konzept der Ontologien ist bereits darauf ausgelegt, eine hohe Interoperabilität zu ermöglichen. Durch Ontologien höherer Stufe wird es ermöglicht, das Wissen aus anderen Ontologien automatisch dem Kontextmodell hinzuzufügen und zudem gibt es zahlreiche, z.B. auf XML basierende, Schnittstellen bei den meisten Triplestores, die das Hinzufügen von Kontextdaten in anderen Formaten ermöglichen.

Logikbasierte Systeme weisen in der Regel keine hohe Interoperabilität auf, da die Kontextinformationen in Fakten und Regeln übersetzt werden müssen, um sinnvoll in die Wissensbasis aufgenommen werden zu können. Entsprechend muss, um Interoperabilität zu erreichen, jeweils ein Modul entwickelt werden, das aus dem anderen Modell solche Fakten extrahieren kann.

Der modellgetriebene Ansatz unterstützt inhärent keine Interoperabilität. Das Kontextmodell besteht hier aus Klassen und ausführbaren Code. Um mit anderen Programmen zusammenzuarbeiten, muss bei diesem Ansatz für jede andere Anwendung eine passende Schnittstelle per Hand geschrieben werden, wobei auch in diesem Fall nur Kontexte unterstützt werden können, die bereits der Anwendung als Klasse bekannt sind.

Einfachheit Um den ontologiebasierten Ansatz mit all seinen Möglichkeiten zu verwenden, ist es notwendig, sich eine Ontologieumsetzung, eine passende Ontologiedatenbank und eine Inferenzengine auszusuchen, sich jeweils in die Dokumentation einzulesen und die entsprechenden Schnittstellen sinnvoll in die Architektur der Anwendung zu integrieren. Vorgefertigte Architekturen und Modelle, wie CoBrA und SOUPA können diese Schritte zwar vereinfachen und beschleunigen, dennoch ist dadurch schon in der Entwurfsphase der Anwendung ein erheblicher Aufwand verbunden.

Logikbasierten Systemen per se wohnt bereits ein sehr hohes Maß an Formalität inne. Um die relevante Kontextdomäne logisch sinnvoll zu modellieren und die Regeln und Fakten in sich stimmig abzubilden, ist im Allgemeinen ein hoher Aufwand an Analyse und Abstraktion notwendig. Zudem sind gute Kentnisse in logischen Programmiersprachen, sowie formaler Logik sinnvoll, um Korrektheit des logischen Systems sicherzustellen. Vereinfachte Formen, wie der regelbasierte ECA-Ansatz können entsprechend auch einfacher zu verwenden sein, erfordern jedoch ebenfalls einiges Einarbeiten in die entsprechenden Architekturen.

Die modellgetriebene Entwicklung erfordert, abgesehen von Kenntnissen in Modellierungssprachen wie UML, nur wenig spezifisches Wissen. Ein gegebenes Metamodell, wie durch ContextUML gegeben kann relativ einfach und schnell verstanden und als Grundlage für die Modellierung verwendet werden. Das genaue Modellieren der Abläufe und Zusammenhänge zu einem Format, das von Codegeneartoren verwendet werden kann, sowie die Verwendung und Konfiguration dieser Generatoren, erfordern ein bisschen Übung, sollten jedoch von einem erfahrenen Softwareentwickler ohne größere Probleme zu bewältigen sein.

Automatische Inferenz Da in der semantischen Beschreibung der Instanzen einer Ontologiedatenbank auch Regeln über Relationen und Klassen enthalten sind, ist es möglich, mit Hilfe dieser Regeln neue Aussagen über den Datenbestand zu schlussfolgern. Mit Hilfe einer Inferenz-Engine kann damit dem Entwickler einiges an Programmierarbeit durch Modellierung in der Wissensbasis abgenommen werden.

Die Möglichkeit zur automatischen Inferenz ist einer der Hauptvorteile der Benutzung des logikbasierten Ansatzes. Durch Anwenden der definierten Regeln kann das Verarbeiten der Kontextinformationen, sowie die Steuerung der ganzen Anwendung erfolgen. Die Inferenz geschieht hier schneller und effizienter als bei Ontologien und sorgt dafür, dass eine gute logische Modellierung eine große Menge Implementierungslogik ersetzen kann.

Die modellgetriebene Entwicklung bietet inhärent keine Möglichkeiten, automatische aus Kontextdaten Schlussfolgerungen zu ziehen. Alles was mit diesem Ansatz abgeleitet werden soll, muss von Hand modelliert bzw. implementiert werden.

4.3 Zusammenfassung des Vergleichs

In Tabelle 1 wird der vorangegangene Vergleich der beschriebenen Ansätze zusammengefasst. Hierbei wird eine Skala von sehr gering ($--$) über neutral (o) bis sehr hoch ($++$) verwendet, um auszudrücken, inwiefern die einzelnen Ansätze diese Kriterien erfüllen.

Tabelle 1. Vergleich der Ansätze

	Ressourcen	**Wiederverw.**	**Flexibilität**	**Interop.**	**Einfachheit**	**Inferenz**
Ontologie	$--$	$++$	$++$	$++$	$-$	$+$
Logik	o	o	$++$	o	$--$	$++$
Modell	$++$	o	$-$	$--$	$++$	$--$

Die Tabelle verdeutlicht, dass sich insbesondere der ontologiebasierte und der modellgetriebene Ansatz in Bezug auf ihre Anwendbarkeit sehr komplementär gegenüber stehen. Komplexe und verteilte Systeme bei denen die Ressourcensparsamkeit und Einfachheit eher im Hintergrund stehen, können stark von den Vorteilen des ontologiebasierten Kontextmodells profitieren, während die modellgetriebene Entwicklung insbesondere in den Fällen zu bevorzugen ist, in denen nur wenige Ressourcen auf der Zielplattform vorhanden sind, oder die Entwickler Wert auf eine einfache Implementierung legen. Der logische Ansatz ist im Vergleich zu den anderen beiden als ein Mittelweg zu sehen. Durch die wesentlich effizienteren und leichtgewichtigeren Inferenz-Engines im Vergleich zum Ontologie-Ansatz ist dieser Ansatz als wesentlich ressourcensparender zu bewerten und entsprechend auch gut ohne Verteilte Client-Server-Architektur zu denken, kann jedoch nicht mit der hohen Wiederverwendbarkeit und Interoperabilität der Ontologien mithalten. Im Vergleich zur modellgetriebenen Entwicklung auf der anderen Seite, zeichnet sich der logische Ansatz in erster Linie durch die Möglichkeit zur automatischen Inferenz, sowie der höheren Flexibilität jedoch auf Kosten der Einfachheit aus. Bei der Wahl des Kontextmodells aus den drei vorgestellten Modellen ist es daher notwendig, eine Abwägung zwischen den beschriebenen Anforderungen zu treffen und somit zu überlegen, welche der Anforderungen für die Anwendung zu priorisieren sind.

4.4 Zusammenfassung und Ausblick

In dieser Ausarbeitung wurden die Begriffe Kontext und Kontextsensitivität definiert und erklärt, sowie ein Überblick über die verschiedenen Möglichkeiten gegeben, Kontext allgemein zu Modellieren und in Architekturen umzusetzen. Anschließend wurde insbesondere der ontologiebasierte Ansatz, der logische Ansatz, sowie die modellgetriebene Entwicklung genauer Beschrieben und anhand von Beispielen erläutert. Abschnitt 4 vergleicht die betrachteten Ansätze basierend auf für den Entwickler relevanten Anforderungen an kontextsensitive Anwendungen und zeigt hierbei die Vor- und Nachteile der einzelnen Ansätze auf.

Wie bei den meisten Übersichten über ein komplexes Thema ist diese Arbeit zwar inhaltsreich jedoch unvollständig. Um eine passende Wahl des Kontextmodells für eine zu entwickelnde Anwendung zu treffen, reichen die vorgestellten Modelle nicht immer aus, weswegen auch andere Ansätze, wie zum Beispiel in [20] vorgestellt, in Betracht gezogen werden sollten. Zudem ist zu erwarten, dass in Zukunft weitere mögliche Modellierungen und Vorgehensweisen entwickelt werden, die auf eine ähnliche Art und Weise evaluiert werden sollten.

Literatur

1. https://code.google.com/p/rdfonthego/.
2. Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.
3. Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.
4. Peter J Brown. Triggering information by context. *Personal Technologies*, 2(1):18–27, 1998.
5. Peter J Brown, John D Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE*, 4(5):58–64, 1997.
6. Guanling Chen, David Kotz, et al. A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.
7. Harry Chen, Tim Finin, and Amupam Joshi. Semantic web in the context broker architecture. Technical report, DTIC Document, 2005.
8. Harry Chen, Tim Finin, and Anupam Joshi. An intelligent broker for context-aware systems. In *Adjunct proceedings of Ubicomp*, volume 3, pages 183–184, 2003.
9. Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2003.
10. Harry Chen, Tim Finin, and Anupam Joshi. The soupa ontology for pervasive computing. In *Ontologies for agents: Theory and experiences*, pages 233–258. Springer, 2005.
11. Patricia Dockhorn Costa, João Paulo A Almeida, Luís Ferreira Pires, and Marten van Sinderen. Evaluation of a rule-based approach for context-aware services. In

Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, pages 1–5. IEEE, 2008.
12. Laura Daniele, Patrícia Dockhorn Costa, and Luís Ferreira Pires. Towards a rule-based approach for context-aware applications. In *Dependable and Adaptable Networks and Services*, pages 33–43. Springer, 2007.
13. Anind K Dey. Context-aware computing: The cyberdesk project. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, 1998.
14. Maja D'Hondt. *Hybrid aspects for integrating rule-based knowledge and object-oriented functionality*. PhD thesis, Citeseer, 2004.
15. Jens Dietrich. The mandarax manual. *Institute of Information Sciences & Technology, Massey University, New Zealand*, 2003.
16. Patrícia Dockhorn Costa. *Architectural support for context-aware applications: from context models to services platforms*. University of Twente, 2007.
17. Ernest Friedman-Hill. *JESS in Action*. Manning Greenwich, CT, 2003.
18. Tom Gruber. Ontology., 2009.
19. Richard Hull, Philip Neaves, and James Bedford-Roberts. Towards situated computing. In *Wearable Computers, 1997. Digest of Papers., First International Symposium on*, pages 146–153. IEEE, 1997.
20. Georgia M Kapitsaki, George N Prezerakos, Nikolaos D Tselikas, and Iakovos S Venieris. Context-aware service engineering: A survey. *Journal of Systems and Software*, 82(8):1285–1297, 2009.
21. George N Prezerakos, Nikolaos D Tselikas, and Giovanni Cortese. Model-driven composition of context-aware web services using contextuml and aspects. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 320–329. IEEE, 2007.
22. Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32, 1994.
23. Quan Z Sheng and Boualem Benatallah. Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In *Mobile Business, 2005. ICMB 2005. International Conference on*, pages 206–212. IEEE, 2005.
24. Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004.

Herausforderungen im Softwareentwicklungsprozess mobiler B2B-Anwendungen im Vergleich zur klassischen Software

Ubiquitäre Systeme Seminar SS2013

Lorenz Haas and Advisor: Nadezda Sackmann[1,2]

[1] Karlsruhe Institute of Technology (KIT), Pervasive Computing Systems - TECO
lorenz.haas@student.kit.edu
[2] nsackmann@googlemail.com

Zusammenfassung Das Entwickeln mobiler Anwendungen für den B2B-Bereich ist aktuell eine herausfordernde Aufgabe, es gibt viele spezielle Anforderungen und technische Herausforderungen bei der Entwicklung der Software für mobile Endgeräte. Diese haben nicht nur Auswirkungen auf die Programmierung, sondern auf den gesamten Entwicklungsprozess. In dieser Seminararbeit werden die unterschiedlichen Entwicklungsmodelle der klassischen und mobilen Softwareentwicklung vorgestellt und miteinander, sowie auf die im Prozess auftretenden Herausforderungen eingegangen.

Key words: mobile Anwendungen, Softwareentwicklungsprozess, Agile Softwareentwicklung, Entwicklungsmodelle, SCRUM, B2B

1 Einleitung und Motivation

Die rasante Entwicklung der letzten Jahrzehnte im Bereich der Informations- und Kommunikationstechnologien sorgen weiterhin für eine anhaltende Beliebtheit von mobilen Endgeräten, wie Smartphones und Tablet-Computer. Darüber hinaus sinkt die Nachfrage nach klassischen Desktop-Computern und Notebooks[12]. Die Verbreitung von günstigen, leistungsfähigen Tablet-Computern und Smartphones beschleunigt die Verlagerung von stationären Desktop-Computern hin zu mobilen Endgeräten. Die mobilen Endgeräte werden überwiegend noch als Ergänzungsgerät zu einem Desktop-Computer gesehen, besonders im Business Bereich bei dem der Fokus auf Sicherheit, Rechnerleistung, einfache Texteingabe und übersichtlicher Darstellung liegt. Jedoch entwickeln sich die mobilen Endgeräte mit steigender Rechnerleistung und der Unterstützung von Peripheriegeräten, hin zu Hauptrechnern. Zu diesem Ergebnis kommt auch Gartner, Anbieter von Marktforschungsergebnissen und Analysen über die Entwicklung in der IT. Nutzer verbringen mehr Zeit mit ihren Tablet-Computern und Smartphones, als mit

ihren Desktop-Computern. Für das Abrufen von E-Mails, oder auch das Surfen im Internet wird eher ein mobiles Endgerät genutzt. Dies führt zu sinkenden Absatzzahlen auf dem PC Markt. Dies ist auch kein temporärer Trend, sondern reflektiert einen anhaltenden Wechsel des Nutzerverhaltens [12]. Tabelle 1. zeigt die von Gartner geschätzten weltweiten Auslieferung von Desktop-Computern und mobilen Endgeräten bis in das Jahr 2017.

Tabelle 1. Prognostizierte, weltweite Auslieferung von PCs und mobilen Endgeräten (Tsd. Stück) nach Gartner [12].

Device Type	2012	2013	2014	2017
PC (Desk-Based and Notebook)	341.263	315.229	302.315	271.612
Ultramobile	9.822	23.592	38.687	96.350
Tablet	116.113	197.202	265.731	467.951
Mobile Phone	1.746.176	1.875.774	1.949.722	2.128.871
Total	2..213.373	2.411.797	2.556.455	2.964.783

Diese Entwicklung sorgt auch für eine steigende Nachfrage und intensive Nutzung der mobilen Anwendungen. Allein in Apples AppStore wurden bereits insgesamt 50 Milliarden Downloads von Anwendungen gezählt. Der Store bietet eine Auswahl von mehr als 850.000 Apps für iOS und davon sind 350.000 speziell für das iPad optimiert [5]. Aber auch für die anderen mobilen Systeme wie z.B. Android gibt es mehrere Hunderttausend Apps im Google Play Store. Die Anwendungen finden aber nicht nur im B2C Bereich (Business-to-Consumer) großen Gefallen, sondern auch die Unternehmen entdecken die Vorteile der mobilen Anwendungen für sich. Die Anzahl der Unternehmensrelevanten B2B Anwendungen (Business-to-Business) haben sich im Zeitraum von 2011 bis 2012 von 100.000 auf 200.000 verdoppelt [6]. Die Kundenerwartungen der Unternehmen an die neuen Technologien sind hoch, diese sollen z.B. nicht nur die Unternehmensprozesse verbessern und die Zusammenarbeit zwischen den Mitarbeitern optimieren, sondern es sollen auch die Anwendungen flexibler und agiler gestaltet werden um somit für kürzere Reaktionszeiten zu sorgen[27]. Der Vorteil beim Einsatz von mobilen Endgeräten ist, dass diese im Vergleich zu Desktop-Computern immer und überall einsatzbereit sind, des Weiteren ist deren Bedienung leicht und intuitiv, sie bieten rund um die Uhr einen Internetzugang und somit z.B. die Möglichkeit von Video-Telefonie unter mehreren Mitarbeitern die sich an unterschiedlichen Orten aufhalten, die Buchung eines Videokonferenzraumes wird dadurch überflüssig und die Kommunikation effektiver[27]. Die Mitarbeiter können schnell und mobil auf Firmendaten und Anwendungen des Unternehmens zugreifen, getreu dem Motto "Anytime, Anywhere, Anything and Anyhow". Die mobilen Endgeräte und ihre Anwendungen verändern somit die Geschäftswelt nachhaltig.

Dieser radikale Wandel in der mobilen Nutzung von Soft- und Hardware beeinflusst auch den Entwicklungsprozess. Dieser hat in den letzten Jahren ebenfalls einen intensiven Wandel vollzogen, um auf die neuen Herausforderungen

und Anforderungen im mobilen Bereich geeignet reagieren zu können. Wo der Schwerpunkt bei klassischer Software noch eher auf Sicherheit und Performance liegt, sind die Herausforderungen der mobilen Entwicklung eher die fehlende Interoperabilität der verschiedenen mobilen Plattformen. Jedoch bieten die mobilen Anwendungen den Unternehmen neue Möglichkeiten, innovative Anwendungen werden die Geschäftsprozesse weiter "mobilisieren"[27]. Denn die Geschäftsabläufe zwischen Unternehmen und Kunden ändern sich sehr stark. Der Kunde wir immer mehr und direkter in die Geschäftsabläufe einbezogen, der Informationsabruf von Unternehmensdaten kann von unterwegs durch mobile Endgeräte erfolgen, dies spart Zeit und ist sehr flexibel. Dies bedeutet aber auch, dass bei der Entwicklung stark auf die Bedürfnisse der Kunden eingegangen werden muss und die Kunden frühzeitig in den Entwicklungsprozess miteinbezogen werden.

Diese Seminararbeit verschafft einen Überblick über das Vorgehen und die jeweiligen Herausforderungen bei der Softwareentwicklung von mobilen Anwendungen im Vergleich zu der klassischen Softwareentwicklung im Unternehmensbereich. Zunächst werden im Grundlagenkapitel, wichtige Begriffe definiert und auf die unterschiedlichen Technologien der App-Entwicklung eingegangen. Anschließend werden im dritten Kapitel unterschiedliche Entwicklungsmodelle der klassischen Softwareentwicklung vorgestellt. Im vierten Kapitel, dem Hauptteil, wird zunächst auf Entwicklungsmodelle eingegangen, die sich besonders für mobile Anwendungen eignen. Anschließend werden die Herausforderungen im Entwicklungsprozess betrachtet. Im Detail geht es um die Hardware- und Umweltbesonderheiten, die mobile Endgeräte mit sich bringen. Im fünften Kapitel werden die Eigenschaften und Herausforderungen klassischer Software und mobiler Software direkt gegenübergestellt und verglichen. Das letzte Kapitel fasst die Arbeit kurz zusammen und bietet einen Ausblick.

2 Grundlagen

Der Begriff mobile Anwendung wurde bisher noch nicht klar definiert, es lassen sich jedoch mehrere Gruppen mobiler Anwendungen hinsichtlich ihres technologischen Hintergrundes unterscheiden. Die Seminararbeit orientiert sich im Folgenden an der Beschreibung des Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM) [8] und [27]. Den größten Marktanteil nehmen native Anwendungen oder Umgangssprachlich auch Apps genannt ein, danach folgen die mobilen Websites oder Umgangssprachlich auch WebApps. Die WebApps gewinnen aktuell an Bedeutung und zeichnen sich durch ihre große Flexibilität aus. Die Grenze zwischen den nativen Apps und den WebApps verschwindet immer mehr und somit lassen sich weitere Gruppen definieren, wie Hybrid Apps, Cross Compiler Apps und Applikationsbeschreibungssprachen. Abbildung 1. soll einen Überblick über die unterschiedlichen Technologiegruppen und Multiplattformumgebungen geben und diese anhand ihrer Flexibilität und Plattformnähe einteilen.

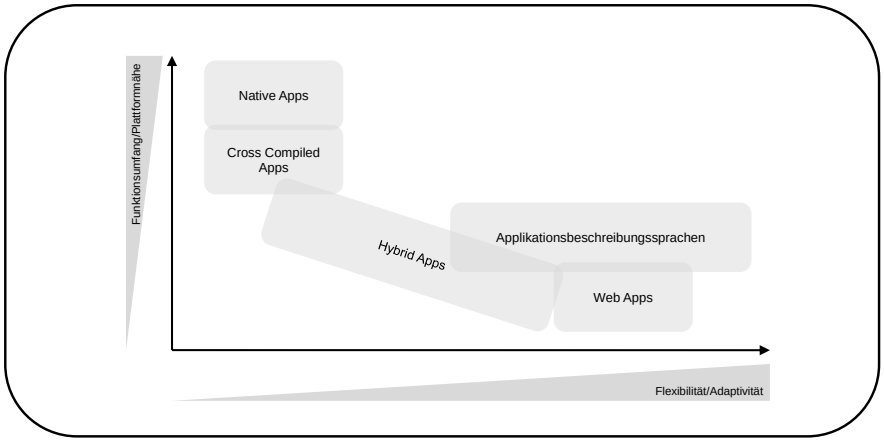


Abbildung 1. Übersicht der unterschiedlichen Gruppen mobiler Anwendungen[27].

2.1 Native Anwendungen

Native Anwendungen werden Umgangssprachlich auch Apps genannt, sie sind kleine Softwareprogramme, die zunächst über einen AppStore heruntergeladen werden müssen und sich anschließend auf Smartphones oder Tablet-Computer installieren lassen. Sie werden speziell für ein System/Geräteklasse wie z.B. Smartphones mit Android oder dem iPad mit iOS entwickelt. Sie bieten z.B. Zugriff auf soziale Netzwerke, E-Mail Konten oder aktuelle Nachrichten. Ein großer Vorteil der Apps ist die Performance, da sie speziell für das mobile Endgerät entwickelt werden. Somit haben die Anwendungen meist direkten Zugriff auf die vom Gerät angebotenen Schnittstellen, z.B. der Kamera, Ortungssensoren, Lagesensoren oder auch auf das Adressbuch. Dies ist vor allem bei Anwendungen, die viel Rechenleistung benötigen wie z.B. Spiele nützlich. Ein großer Nachteil ist, dass kaum Synergieeffekte bei der Entwicklung für mehrere Systeme genutzt werden können und somit meist höhere Entwicklungskosten anfallen. Es muss für die unterschiedlichen Systeme jeweils eine eigenständige App entwickelt werden, hinzukommt auch die Schwierigkeit der Hardware- bzw. Softwaresystem-Fragmentierung. Im Hardwarebereich sind dies die unterschiedlichen Geräteklassen wie Smartphone und Tablet-Computer, die sich wiederum in der Bildschirmgröße und Funktionsumfang der Geräte unterscheiden. Bei Geräten von Apple ist dies noch recht überschaubar, da dies ein geschlossenes System ist und Hardware nur von Apple hergestellt wird (iPhone 1-4, iPhone 5, iPad 1-4, iPad mini). Android hingegen ist ein freies, quelloffenes System. Es gibt viele unterschiedliche Geräte von unterschiedlichen Herstellern. Des Weiteren sind auch noch viele, unterschiedlichen Softwaresystem-Versionen verbreitet, bei Android werden momentan ungefähr 6 Versionen noch unterstützt (Eclair, Froyo, Gingerbread,...)[19]. Tabelle 2. gibt einen Überblick über die unterschiedlich verwendeten Programmiersprachen und Entwicklungsumgebungen für die jeweiligen Systeme.

Tabelle 2. Entwicklungstools und Sprachen zur Entwicklung nativer Anwendungen für die unterschiedlichen Plattformen [27].

Betriebssystem	Programmiersprache	Entwicklungsumgebung
iOS	Objective-C	Xcode
Android	Java	Eclipse
Windows Phone 7	C#, Visual Basic	Visual Studio
BlackBerry	Java	Eclipse
Symbian	Java	Symbian IDE

Hierdurch wird deutlich wie unterschiedlich die Systeme sind. Als verbreitetste Programmiersprache gilt Java, jedoch müssen für die unterschiedlichen Systeme deren eigene Frameworks genutzt werden, um die jeweiligen systemspezifischen Charakteristika abzubilden. Des Weiteren haben die einzelnen Systeme auch eigenen Programmierrichtlinien[17][18][21].

2.2 Mobile Websites

Mobile Websites werden Umgangssprachlich auch WebApps genannt. Dies sind Anwendungen die in erster Linie im Browser des jeweiligen Gerätes laufen. Die Daten werden über einen Webserver bereitstellen und sind somit plattformunabhängig. Sie werden mit Hilfe von Webtechnologien wie HTML, JavaScript und CSS erstellt und sind meist für die kleinen Bildschirme der mobilen Endgeräte optimiert, die Inhalte und Nutzernavigation passt sich entsprechend an. Für deren Nutzung ist keine Installation über einen App Store nötig, sondern die relevanten Informationen werden vom Webserver geladen. Des Weiteren lassen sich die Daten auf dem Webserver schnell und einfach aktualisieren und es muss nicht wie bei einer nativen Anwendung eine neue Version heruntergeladen werden. Ein Nachteil ist dass eine WebApp nur begrenzt Zugriff auf die Hardwareschnittstellen besitzt und die Performance bei Rechenintensiven oder Grafisch aufwendigen Anwendungen sehr darunter leidet. Abhilfe schaffen hierbei jedoch Frameworks wie z.B. jQuery Mobile, [11] dadurch lässt sich die Entwicklung von mobilen Webapplikationen erleichtern und das "Look and Feel"lässt sich an denen einer nativen Anwendung anpassen. Des Weiteren lässt sich durch Frameworks auch die Wiederverwendbarkeit von Programmcode verbessern. Die WebApps lassen sich schnell und einfach erstellen, besonders mit vorhandenen Frameworks. Außerdem sind Änderungen schnell erledigt es müssen lediglich die Daten auf dem Server aktualisiert werden, es entfällt somit der Update-Prozess, wie man ihn bei nativen Apps kennt. Das macht die WebApps sehr flexibel und für Unternehmens Anwendungen interessant.

2.3 Hybrid Apps

Die Hybrid Apps nutzen die Vorteile von nativen Apps und WebApps. Hierbei werden die HTML, CSS und JavaScript Dateien als Ressource genommen und in

einer nativen App eingebettet. Diese kann dann ganz normal über den jeweiligen App Store vertrieben werden. Es lässt sich somit sehr einfach eine Anwendung für mehrere Plattformen entwickeln. Der Zugriff auf einen Webserver wird optional, indem man die notwendigen Daten der Anwendung lokal mit der App ausliefert und auf dem Gerät speichert. Die zwei bekanntesten Frameworks hierfür sind Phonegap[2]und Titanium[4]. Für einfache Anwendungen ist dies eine gute Lösung, die Apps lassen sich somit schnell und einfach für die unterschiedlichen Systeme entwickeln und man erkennt kaum ein Unterschied im Verhalten zu nativen Apps. Jedoch sind Hybrid Apps für rechenintensive Anwendungen weniger geeignet.

PhoneGap Phonegap ist ein Framework durch das sich native Anwendungen mit Hilfe der Webtechnologien HTML5, JavaScript und CSS für mobile Endgeräte und deren unterschiedliche Betriebssysteme wie iOS, Android, Windows Mobile, usw. erstellen lassen. Diese Anwendungen könne dann in den App Stores der unterschiedlichen Systeme vermarktet werden. Der große Vorteil dabei ist, dass nur eine Webanwendung erstellt werden muss und diese für die unterschiedlichen Systeme leicht angepasst werden kann. Dies senkt die Entwicklungskosten erheblich[2].

Titanium Titanium ist ein Framework ähnlich wie PhoneGap, jedoch versucht es zur Laufzeit native Objekte zu generieren um z.B. das User-Interface des jeweiligen Gerätes zu nutzen. Dabei wird der JavaScript Code analysiert und in plattformspezifische, ausführbare Elemente übersetzt. Es stellt daher ein deutlich größeres API für die systemnahen Funktionen zur Verfügung, dazu gehören Geodaten, Beschleunigungssensor, Kamera, Schnittstellen zu Facebook usw. Ein Nachteil von Titanium ist, das es zur Zeit lediglich zwei Plattformen unterstützt dazu gehören iOS und Android. Auch hier können anschließend die Anwendungen in den jeweiligen App Stores vermarktet werden[4].

2.4 Cross Compiled Apps

Diese Art von Apps richtet sich weniger an die Webentwickler, sondern unterstützt auch andere als die jeweils vorgeschriebenen Programmiersprachen und Programmierumgebungen, welche in Tabelle 2 dargestellt wurden. Somit lassen sich z.B. auch mit Microsofts .NET Anwendungen für iOS oder Android entwickeln. Ein Vertreter der Cross-Plattform ist MonoTouch[29].

MonoTouch Als Programmiersprache wird C# verwendet, ähnlich wie bei den Hybrid Apps wird hier von MonoTouch APIs für die systemnahen Funktionen zur Verfügung gestellt. Des Weiteren ist es möglich die nativen Bibliotheken der jeweiligen Zielplattform einzubinden. Die fertige Anwendung kann in den jeweiligen App Stores vermarktet werden[29].

2.5 Applikationsbeschreibungssprachen

Bei Applikationsbeschreibungssprachen (ADL) werden anders wie bei mobilen Websites keine Webtechnologien eingesetzt, sondern eine eigens für die Smartphone-Systeme konzipierte Sprache, wie das App Construction Kit[13]. Hierbei wird das User-Interface sehr genau beschrieben, jedoch sehr abstrakt über die unterschiedliche Systeme hinweg. Dadurch lässt sich eine Anwendung für mehrere Systeme entwickeln und wirkt auf der jeweiligen Zielplattform sehr nativ.

Für den Entwicklungsprozess ist es also sorgfältig abzuwägen welche Variante von mobilen Anwendungen für den jeweiligen Einsatzzweck genutzt werden sollte. Da jede der Varianten sowohl Vor- als auch Nachteile hat. Jedoch geht der Trend laut einer BITKOM-Umfrage im B2B-Bereich in Richtung WebApps, da sich Inhalte flexibler transportieren und anpassen lassen[8]. Außerdem bieten plattformunabhängige Architekturen den Vorteil, dass eine Anwendung nur einmal für die unterschiedlichen Systeme erstellt werden muss, dies senkt die Entwicklungskosten enorm. Für Anwendungen mit hohen Anforderungen an die Leistung bieten sich native Anwendungen an. Es kommt aber auch auf die Zielgruppe und deren Einsatzzweck an. Dies ist eine weitere Möglichkeit, mobile Anwendungen zu gruppieren. Es lässt sich Unterscheiden zwischen B2C Anwendungen, die für den Endkunden gedacht sind und B2B Anwendungen die speziell für Unternehmen entwickelt werden.

2.6 B2C Anwendungen

Unter B2C Anwendungen versteht man Business to Consumer Anwendungen, hierbei steht der Endkunde im Fokus. Dies sind z.B Spiele wie Angry Birds, oder auch Apps der sozialen Netzwerke wie Facebook, welche man aus den App Stores laden kann. Hier dominieren native Anwendungen den Markt, da die Anwendungen meist eine hohe Rechenleistung fordern wie z.B. für Spiele mit 3D Grafik. Sie lassen sich auch leichter über die App Stores vermarkten.

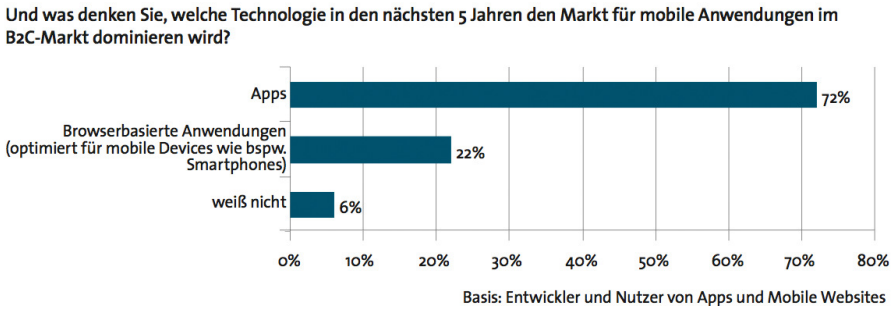


Abbildung 2. Umfrage der BITKOM an Entwickler und Nutzer[8].

Im Mittelpunkt der Seminararbeit stehen B2B Anwendungen, welche speziell für den Unternehmensbereich entwickelt werden.

2.7 B2B Anwendungen

Unter B2B Anwendungen versteht man Business to Business Anwendungen, hierbei steht das Unternehmen im Fokus. Sie werden speziell für den Einsatz im Betrieb konzipiert, um z.B. den Verkauf, Vertrieb oder den Kundenservice zu unterstützen. Der Großteil der aktuell verfügbaren Anwendungen beschäftig sich jedoch noch mit Office-, Produktivitäts- oder Kollaborationswerkzeuge oder Analyse-Tools, welche Kennzahlen aus betrieblichen Systemen bereitstellen. Jedoch sind Apps, die umfangreiche Geschäftsabläufe darstellen bisher nicht allzu sehr verbreitet. Die B2B Anwendungen sind meist sehr anwenderorientiert und beschäftigen sich mit dem Wesentlichen. Sie sollten einfach zu bedienen und grafisch ansprechend aufbereitet sein. Im B2B-Bereich sind WebApps sehr interessant, da diese flexibler sind und sich die Entwicklungskosten für mehrere unterschiedlichen Systemen in Grenzen halten.

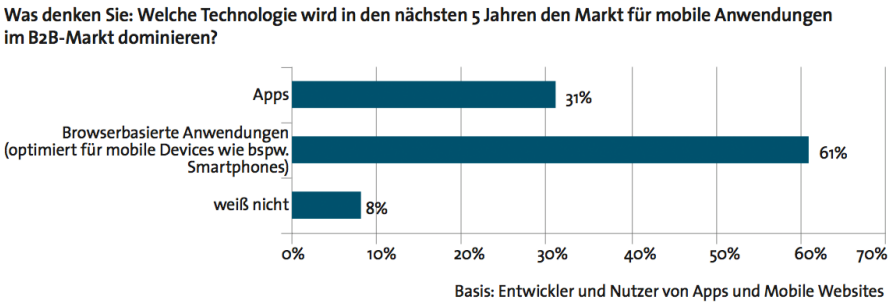


Abbildung 3. Umfrage der BITKOM an Entwickler und Nutzer[8].

Es existiert ein riesiges Angebot von mobilen Business Anwendungen und diese entsprechen im Prinzip jenen für Desktop-Computern. Wobei die mobile Variante nie das gesamte Spektrum abdeckt, sondern lediglich einen Teilbereich. Meist werden mobile Business Anwendungen nur zur Unterstützung der Desktop-Variante verwendet. Im folgenden sollen nun ein par Beispiele für B2B-Anwendungen gegeben werden.

SAP Travel Expense: Diese App hilft die Reisekosten zu verwalten, bei den meisten Systemen zur Reisekosten- und Spesenabrechnung ist ein Login über einen Desktop-Computer nötig. Die App ermöglicht es jedoch unabhängig vom aktuellen Standort, die Kosten direkt einzutragen und erspart somit das Nachtragen zu einem späteren Zeitpunkt. Es lassen sich somit Belege vor Ort z.B. aus einem Restaurant abfotografieren und übermitteln. Dies spart dem

Arbeitnehmer viel Zeit und reduziert den Verwaltungsaufwand, somit kann er sich eher um Unternehmensrelevante Aufgaben kümmern[22].

RMPioniers cobra: Dies ist eine komfortable Kontakt- und Adressmanagement Anwendung für Windows Phone 7, Apples iOS und BlackBerry. Dadurch lässt sich schnell, einfach und in Echtzeit auf Kunden- und Vertribesinformationen zugreifen. Die Datensätze wie Adressen oder Termine lassen sich aus der eigenen cobra-Datenbank anzeigen und bearbeiten. Des Weiteren bietet es eine Suche über alle Adressfelder und Stichwörter. Dies vereinfacht den Mitarbeitern die Arbeit unterwegs und spart Zeit.[22].

2.8 App Stores

Die nativen Anwendungen lassen sich aus den jeweiligen App Stores der mobilen Betriebssystem Hersteller beziehen. Bei Apple ist dies der Apple AppStore, bei Android ist es der Google play Store (ehemals Android Market) und bei Microsoft ist es der Windows Store. Der größte Anteil der Anwendungen die dort gefunden werden können beziehen sich jedoch nur auf B2C Anwendungen. Apple hat vergangenes Jahr das "Volume Purchase Program for Business"gestartet, dies ist ein Business Store, in dem B2B App Lizenzen in größeren Volumen für Unternehmen gekauft werden können. Es lassen sich selbstverständlich die Anwendungen welche im Unternehmen selbst entwickelt wurden auch direkt auf die eigenen Unternehmensgeräte installieren. Bei WebApps müssen die Unternehmen sich selbst um die Vermarktung kümmern, die größeren Unternehmen besitzen bereits einen eigenen App Store, indem sie ihre Produkte direkt zum Download anbieten oder auf die App Stores der unterschiedlichen Systemhersteller verweisen[3].

3 Softwareentwicklungsprozess klassischer Software

Der Begriff Software Engineering wurde 1968 auf der NATO-Konferenz in Garmisch-Partenkirchen geprägt, hierbei war das Ziel der Softwareentwicklung eine strukturierte, ingenieursmäßige Vorgehensweise vorzugeben[23]. Das Institute of Electrical and Electronics Engineers (IEEE) definiert das Software Engineering als

> "The systematic, disciplined, quantifiable approach to the development, operation and maintenance of software"[16].

Man fasst darunter aber auch den Prozess, bei dem eine einzelne Person oder ein ganzes Team die Herstellung eines Software Systems von der Idee und dem Konzept bis hin zur Veröffentlichung der Software managet und organisiert. Sogenannte Entwicklungs- oder Vorgehenssmodelle sind dabei definierte Standards, die helfen sollen Projekte erfolgreicher abzuschließen.
Im Folgenden soll das Basismodell der Softwareentwicklung und einige Entwicklungsmodelle vorgestellt werden gemäß[14].

3.1 Basismodell der Softwareentwicklung

Das Basismodell der Softwareentwicklung umfasst den ganzen Prozess von der Idee bis hin zur Inbetriebnahme der Software. Das Modell lässt sich in drei Bereiche gliedern, den Projektmanagement-Bereich, den Kernbereich und den Projektinfrastruktur-Bereich. Diese Art der Aufteilung lässt sich in vielen anderen Ansätzen wiederfinden. Abbildung 4. gibt einen Überblick des Basismodells.

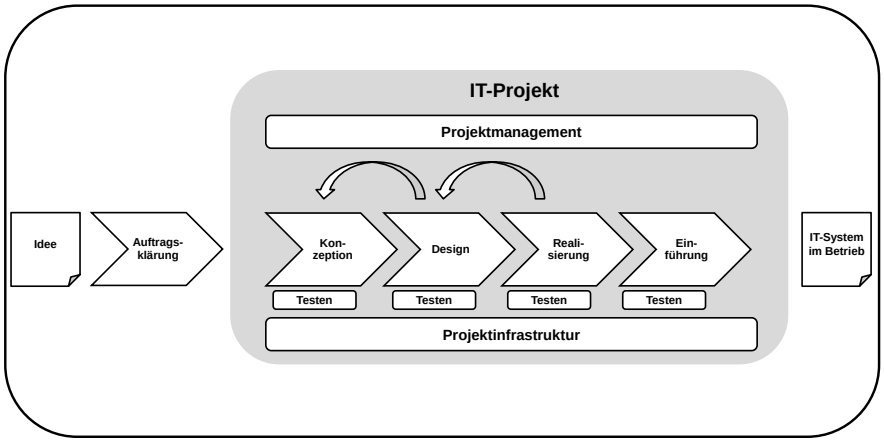


Abbildung 4. Überblick über das Basismodell in Anlehnung an[14].

Alles beginnt mit einer ersten Idee, diese muss dann geprüft und konkretisiert werden. Dieser Vorgang findet in der Auftragsklärungs-Phase statt. Ziel ist es, herauszufinden, ob die Umsetzung der Idee hinsichtlich Wirtschaftlichkeit, Machbarkeit und der Übereinstimmung der firmeninternen Politik für das Unternehmen sinnvoll ist. Das eigentliche IT-Projekt startet mit der Konzeptions-Phase, hier werden die Anforderungen an das System festgelegt. In der Design-Phase wird entschieden wie die Soft- bzw. Hardwarearchitektur des IT-Systems aussehen soll. In der Realisierungs-Phase findet das programmieren statt. Ziel ist es alle Anforderungen, welche in den vorherigen Phasen festgelegt wurden zu erfüllen und die Software fertig zustellen. In der Einführungsphase wird das Produkt beim Kunden eingeführt, das sogenannte "Roll-Out". Hierzu gehört die Abnahme des Systems, die Überführung in den produktiven Betrieb und die Schulung der Anwender. Das Testen findet in jeder der Phasen des IT-Projekts statt, hierdurch sollen Fehler frühzeitig gefunden und entfernt werden. Die Phasen werden nacheinander abgearbeitet, jedoch ist es möglich in der Design- und der Realisierungs-Phase einen Schritt zurückzuspringen.

3.2 Entwicklungsmodelle

Die Idee hinter den klassischen Entwicklungsmodellen der Softwareentwicklung entsprechen denen eines Entwicklungsplans für das Erstellen eines Programms. Mit deren Hilfe gelingt die Zerlegung in einzelne Arbeitsgänge, dies ist wichtig, falls das Projekt im Ganzen für die Entwickler nicht mehr so einfach überschaubar ist. Es werden nun einige der klassischen Entwicklungsmodelle vorgestellt.

Wasserfall-Modell Als das grundlegendste Entwicklungsmodell der Softwareentwicklung gilt das Wasserfall-Modell, es wurde 1970 von Royce dargelegt. Es ist ein sehr einfaches und weit verbreitetes Verfahren. Hierbei werden die einzelnen Phasen strikt nacheinander abgearbeitet und die Ergebnisse jeweils in die nächste Phase weitergetragen. Die nächste Phase kann aber nur beginnen, wenn die vorherige Phase abgeschlossen wurde.

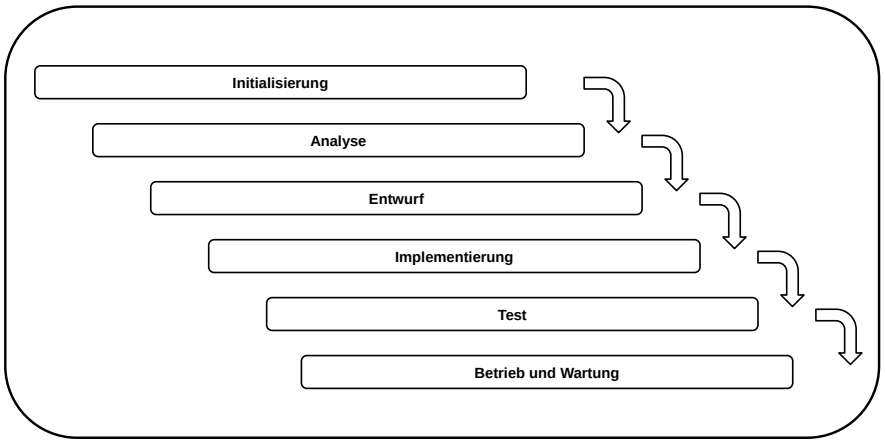


Abbildung 5. Überblick über das Wasserfallmodell in Anlehnung an[14].

Eines der größten Nachteile des Modells, ist dass das Testen als einmalige Aktion am Ende des Entwicklungsprojekts angesiedelt ist. Ein weiterer Nachteil, ist die Anwendbarkeit des Modells bei komplizierteren Projekten. Des Weiteren werden in diesem Modell das Projektmanagement und die Projektinfrastruktur nicht betrachtet und es werden keine Aussagen über Rollen oder Methoden getroffen. Jedoch wird das Modell in verschiedenen Varianten auch heute noch für IT-Projekte genutzt, da es durch seine Einfachheit besticht.

V-Modell Als Erweiterung des Wasserfall-Modells wird das V-Modell von Boehm gesehen, welches in den 1980er erstmals beschrieben wurde. Den Namen hat es auf Grund seiner Struktur, auf der linken Seite befinden sich die unterschiedlichen

Phasen, ähnlich dem Wasserfall-Modell und auf der rechten Seite jeweils passende Tests. Diese beiden Zweige werden durch unterschiedliche Ebenen durch Testfälle miteinander verbunden.

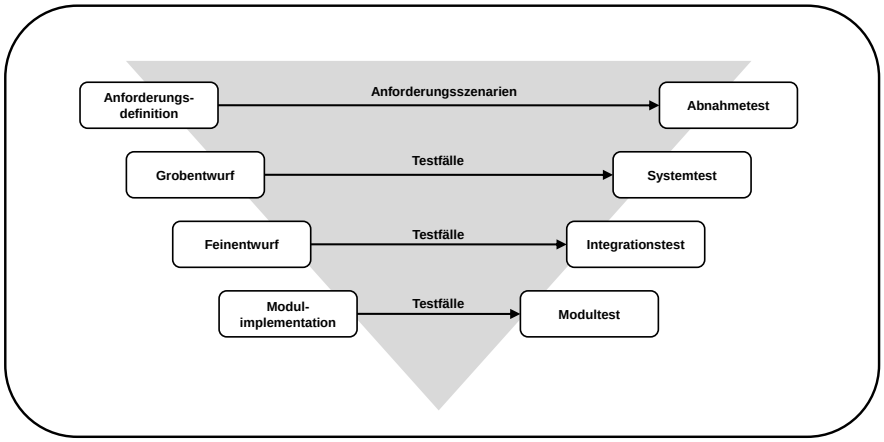


Abbildung 6. Überblick über das V-Modell in Anlehnung an[14].

Hier wurde dem Software-Test eine größere Bedeutung zugeordnet. Das spezielle V-Modell XT Bund, des Bundes und der Länder ist z.B. für den Bereich der Bundesverwaltung verbindlich [9]. Die Anforderungen sind in diesem Modell jeweils sehr genau festgelegt. Aus diesen Gründen, ist das Modell für große Projekte geeignet, für kleinere ist es eher umständlich.

Spiral-Modell Bei dem Spiral-Modell wird das Projekt Spiralförmig durchlaufen und passiert dabei mehrmals in verschiedenen Zyklen unterschiedliche Quadranten:

- Festlegung der Ziele, Randbedingungen und Alternativen
- Bewertung der Alternativen
- Fertigstellung und Abnahme
- Planung des nächsten Zyklus

Im ersten Quadranten werden die groben Gesamtziele, die Rahmenbedingungen und Lösungen festgelegt. Diese werden bei jedem weiteren Zyklus konkretisiert und verfeinert, sowie speziell für die einzelne Systemkomponenten beschlossen. Im zweiten Quadranten werden Alternativen aufgezählt und bewertet. Im dritten Quadranten wird dann die beste Alternative ausgewählt, dann getestet und schließlich umgesetzt. Im letzten Quadranten wird der nächste Zyklus geplant.

Eine der Vorteile des Modells, ist das in jedem Zyklus eine Art Risikoabschätzung stattfindet, welche z.B. Kostenexplosionen verhindert. Somit kann das Projektmanagement frühzeitig Schwächen erkennen und eingreifen.

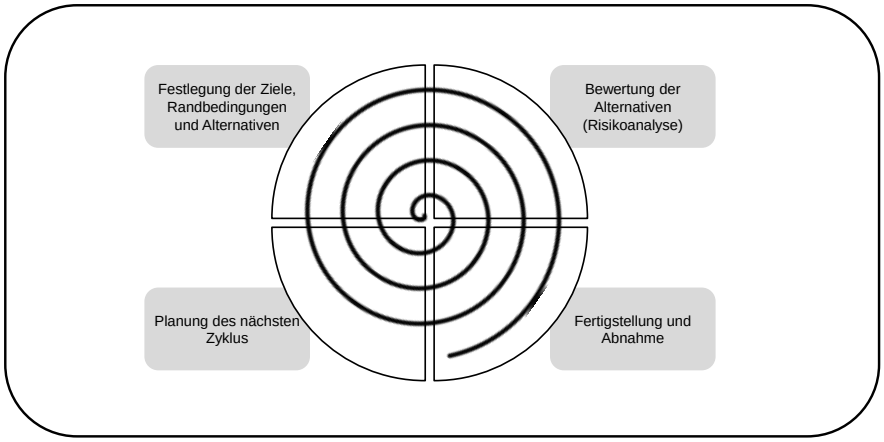


Abbildung 7. Überblick über das Spiral-Modell in Anlehnung an[14].

Extreme Programming Ein etwas anderer, agiler Ansatz, als die zuvor vorgestellten Entwicklungsmodelle ist das Extreme Programming (XP). Dieses entstand in den 1990er Jahren und ist ein Modell bei dem sehr flexibel auf etwaige Änderungen in einem Projekt reagiert werden kann. Es konzentriert sich dabei besonders auf das Programmieren und ist somit sehr flexibel. Man verzichtet gänzlich auf Dokumentationen, da diese zu aufwendig sind und bei jeder Änderung angepasst werden müssen. Dafür wird der Programmcode entsprechend ausführlicher kommentiert. Es gibt keine einzelnen Phasen, sondern es existieren 4 grundlegende Werte.

- Kommunikation
- Einfachheit
- Feedback
- Mut

Es wird großen Wert auf die Kommunikation mit den anderen Mitgliedern des IT-Teams gelegt. Die Lösungen im Programmcode sollten so einfach wie möglich sein. Des Weiteren sollte ständig Feedback gegeben werden, um Missverständnisse frühzeitig auszuräumen, hierunter wird auch das Testen der Software verstanden. Unter dem letzten Punkt versteht man, dass der Entwickler eigenverantwortlich und umsichtig reagieren soll. Es gibt keine Spezialaufgaben, sondern es soll immer der Blick auf das Ganze gewahrt werden. Es wird bei diesem Modell Pair-Programming betrieben, es existieren kurze Entwicklungszyklen, Wiederverwendbarkeit von Programmcode und ein weiterer wichtiger Punkt ist die Kundenbeteiligung, der Kunde wird direkt in das Entwicklerteam integriert.

Nachteil dieses Modells ist, dass es nur für kleinere Projekte praktikabel ist, durch die Ablehnung von einer Dokumentation findet ein Qualitätsverlust statt

und das Pair-Programming ist sehr aufwendig. Jedoch bietet das Modell viele gute und wichtige Ansätze, wie z.B. die Kommunikation und das Feedback geben, sowie das enge Zusammenarbeiten mit dem Kunden.

4 Softwareentwicklungsprozess mobiler Anwendungen

Im vorherigen Kapitel wurden einige Entwicklungsmodell vorgestellt, die den Entwicklern bei der Herstellung von Software unterstützen und ihnen eine strickte Schritt für Schritt Anleitung vorgeben. Dadurch lassen sich Probleme und Herausforderungen frühzeitig erkennen und beheben. Auch die Kommunikation unter den Mitarbeitern lässt sich dadurch verbessern und das Projekt bleibt überschaubar. Viele Punkte der klassischen Softwareentwicklung behalten bei der Entwicklung von mobiler Software natürlich ihre Bedeutung und es werden auch viele grundlegenden Ansätze übernommen, jedoch weist die Entwicklung mobiler Anwendungen eine ganz andere Dynamik auf als die klassische Entwicklung von B2B Software. Die Entwicklungszyklen sind viel kürzer, das ganze Projektsetting ist ein anderes, es muss viel flexibler und agiler entwickelt werden, es gibt ganz andere Anforderungen bezüglich der Hardware oder bei der Vermarktung. Die gesamte Geschäftswelt wird mobilisiert, daher gilt es für die Entwickler Innovationen für die Kunden zu erstellen, die Orts- und Zeitunabhängig sind und mit deren Hilfe auf Daten und Anwendungen zugegriffen werden kann. Dabei lässt sich mit den Anwendungen nicht die gesamte Wertschöpfungskette abdecken, daher gestaltet sich die Einführung der mobilen Anwendungen im B2B-Bereich sehr schwierig. Es muss die Connectivity, das Mobile Device Management und vieles mehr berücksichtigt werden. Die Abbildung 8 bietet einen guten Überblick über die Wertschöpfungskette klassischer und mobiler Unternehmensanwendungen[27].

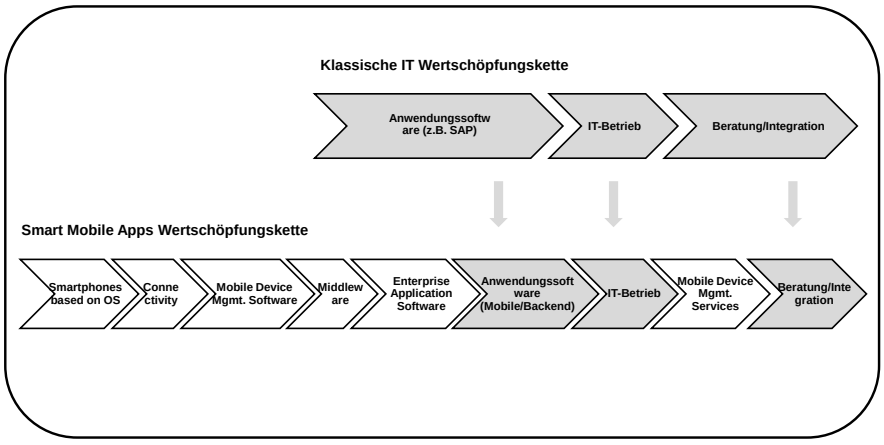


Abbildung 8. Überblick über die Wertschöfpungskette klassischer und mobiler B2B Anwendungen gemäß[27].

4.1 Entwicklungsmodelle

Nachdem im vorherigen Kapitel einige klassische Entwicklungsmodelle behandelt wurden, werden nun zwei Entwicklungsmodelle vorgestellt, die sich speziell in der mobilen Softwareentwicklung einsetzten lassen. Es handelt sich hierbei um agile Softwareentwicklungsprozesse, dem Scrum-Modell und dem mobile-D Modell.[7].

Scrum-Modell Eine agiles und bekanntes Entwicklungsmodell der Softwareentwicklung ist Scrum[20], welches sich auch besonders für den mobilen Einsatz anbietet[25]. Scrum bietet den Entwicklern mehr Handlungsmöglichkeiten und dynamisiert den gesamten Entwicklungsprozess. Es nutzt bei der Entwicklung iterative Zyklen mit fester Länge von 1-4 Wochen, die sogenannten Sprints. Das Entwicklerteam legt dabei selbst fest, welche Aufgaben es wie erledigt. Dabei werden die Anforderungen an die Software in eine Feature-Liste gesammelt, dem Product Backlog. Die daraus entstehenden Aufgaben für die einzelnen Sprints werden im Sprint-Backlog festgehalten. Feedback ist ein wichtiger Bestandteil dieses agilen Ansatzes, es erfolgen tägliche Status-Updates im Daily Scrum-Meeting, dabei wird das bisherige Ergebnis bewertet. Im Idealfall ist ein großer Teil der Entwicklung nach 30 Tagen geschafft und die nächste Runde beginnt. Es gibt drei, gleichberechtigte Rollen, der Product Owner welcher auf der Auftraggeberseite agiert , das Team auf der Entwicklerseite und dazwischen der Scrum-Master, der für Transparenz und gute Arbeitsbedingungen im Projekt zuständig ist.

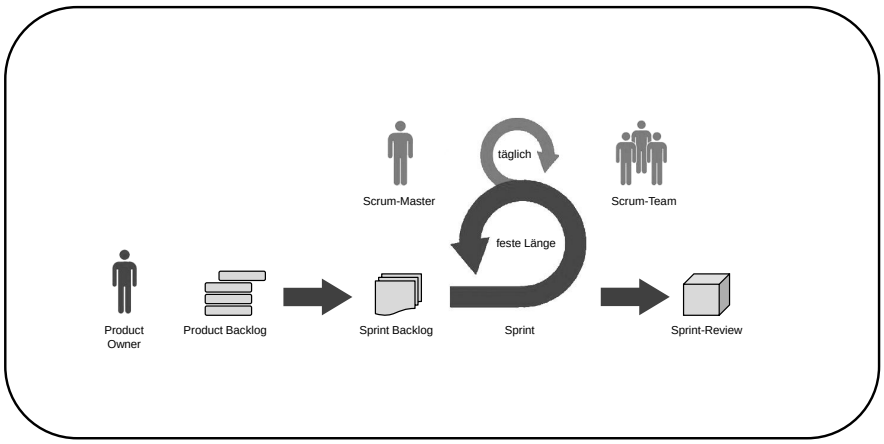


Abbildung 9. Scrum Entwicklungsmodell

mobile-D Mobile-D[1] ist ein agiler Entwicklungsprozess und basiert dabei auf Entwicklungs Praktiken des Extreme Programming, Verfahrensskalierbarkeit der

Crystal Methologie und den Lebenszyklus Reports des Rational Unified Process. Dabei ist der Ansatz auf eine Teamgröße von weniger als 10 Entwickler in einem gemeinsamen Büro ausgelegt. Die Auslieferungszeit beläuft sich bei diesem Ansatz auf einen sehr kurzen Zeitraum von weniger als 10 Wochen.
Dabei ist das Entwicklungsprojekt in fünf Iterationen aufgeteilt, set-up, core, core2, stabiliue und wrap-up. Jede dieser Phasen besteht aus drei unterschiedlichen Arten von Entwicklungstagen Planungs-, Arbeits- und Veröfentlichungs-Tag. Falls unterschiedliche Teams an unterschiedlichen Teilen der Anwendung arbeiten ist noch ein Integrations-Tag nötig. Die Praktiken der unterschiedlichen Phasen umfassen neun grundlegende Elemente:

- Phasing and Pacing
- Architecture Line
- Mobile Test-Driven Development
- Continuos Integration
- Pair Programming
- Metrics
- Agile Software Process Improvement
- Off-Site Customer
- User-Centered Focus

Die Vorteile von Mobile-D in der mobilen Softwareentwicklung sind zum einen, dass die Fortschritte der Entwicklung besser Sichtbar sind, das technische Probleme früher identifiziert und gelöst werden können, es gibt eine gemeinsame Verantwortung der Aufgaben, der Informationsfluss unter den Entwicklern und den Beteiligten ist effizienter, geringe Defektdichte in den veröffentlichten Produkten und ein konstanter Entwicklungsrhythmus.

Vorteil der agilen Softwareentwicklung, ist dass sich diese generell, einfach in Unternehmen anwenden lassen und gut funktionieren. Traditionelle Ansätze sind meist vorhersagbar und lassen sich sehr genau planen. Sie sind sehr Prozessbezogen. Die Organisationsstruktur ist dabei meist mechanistisch, bürokratisch und sehr formell. Agile Ansätze hingegen sind eher für kleinere Teams und setzten auf kontinuierliche Verbesserung, sie sind Mensch-bezogen. Die Organisationsstruktur ist hierbei organisch, flexibel und kooperativ. Außerdem wird die Zusammenarbeit mit dem Kunden gefördert. Die Entwickler konzentrieren sich auf die jeweils aktuelle Aufgabe. [26] Diese Vorteile sind von großer Bedeutung um mit den Herausforderungen, welche bei den unterschiedlichen Phasen der Entwicklung mobiler Anwendungen auftreten können, richtig umgehen zu können. Im folgenden soll nun auf die einzelnen Herausforderungen im Detail eingegangen werden.

4.2 Herausforderungen

Es gibt einige Unterschiede bei der Softwareentwicklung mobiler Anwendungen im Vergleich zu der Entwicklung klassischer Software, die mobilen Endgeräte

weisen spezielle physikalische aber auch technische Eigenschaften und Gegebenheiten auf, die sich auch auf die Anforderungen der Software auswirken und bei der Entwicklung bzw. des Entwicklungsprozesses berücksichtigt werden müssen. Des Weiteren gibt es Besonderheiten in der Umwelt, an die sich der Entwicklungsprozess anpassen muss.

Hardware

- Größe, Gewicht
  Heutige mobile Endgeräte sind so klein, dass sie einfach in die Hosentasche passen. Sie lassen sich somit leicht überall mit hin transportieren und nutzen. Sie sind nicht mehr an einen Ort gebunden wie die Desktop-Computer. Der Nutzer kann somit unterwegs z.B. auf Geschäftsreisen E-Mails schreiben, Dokumente bearbeiten oder Berichte präsentieren. Er hat somit sein Büro mobil immer dabei. Diesen Vorteil gilt es bei der Herstellung von mobilen Anwendungen zu nutzen. Jedoch sind auch geeignete Maßnahmen zu treffen um die Daten vor unrechtmäßigem Zugriffen zu schützen. Denn durch diese gewonnene Mobilität, werden auch vertrauliche Daten aus dem Unternehmen getragen.

- Bildschirmgröße
  Eines der größten Herausforderungen ist es die Informationen geeignet und übersichtlich darzustellen. Auf Desktop-Computern können einfach große Datenmengen übersichtlich auf den großen Monitoren dargestellt werden. Der Tablet-Computer Bildschirm und besonders der Smartphone Bildschirm ist nur ein Bruchteil so groß, sollte aber dennoch in der Lage sein, die gleichen Informationen lesbar und übersichtlich darzustellen. Aus diesem Grund muss bei der Softwareentwicklung darauf geachtet werden, dass man z.B. die enorme Datenmengen nur auf relevante Kennzahlen reduziert, dass unnötige Informationen ausgeblendet werden können oder dass die Informationen in kleiner darstellbaren Gruppen unterteilt werden. Eine wichtige Funktion ist das Zoomen, dadurch lassen sich wichtige Funktionen vergrößern[10].

- Datenübertragung
  Hierbei ist zu beachten, dass nicht jedes mobile Endgerät die Möglichkeit bietet Daten auf das Gerät zu übertragen und umgekehrt. Desktop-Computer haben hierfür CD/DVD/Blu-Ray-Laufwerke, Speicherkartenslots, USB Anschlüsse für USB-Sticks oder externe Festplatten aber auch Netzwerkanschlüsse z.B. für das Firmeninterne Intranet, W-LAN oder Bluetooth. Bei der mobilen Softwareentwicklung ist zu beachten, dass allgemeine Datenübertragungsmöglichkeiten, welche die meisten mobilen Endgeräte unterstützen z.B. über W-LAN, Bluetooth oder Cloud-Dienste über das Internet (EDGE, 3G, HSPA) genutzt werden. Zu beachten ist auch, dass nicht überall eine Internetverbindung verfügbar ist, man aber trotzdem mit der Anwendung arbeiten können sollte. Darum ist es Wichtig, Daten lokal zu speichern[10].

- Erweiterbarkeit
  Die Hardware von Desktop-Computern ist weitaus langlebiger, als mobile Endgeräte da einzelne Komponenten wie z.B. Arbeitsspeicher, Prozessor oder die Festplatte sehr einfach ausgetauscht werden können. Mobile Endgeräte lassen sich meist nicht aufrüsten oder erweitern, es muss dabei auf die nächste Geräteversion gewechselt werden. Dies bedeutet für die Softwareentwickler, dass die Anwendung sehr flexibel sein muss.

- Peripheriegeräte
  Für Desktop-Computer gibt es bereits eine Vielzahl von Peripheriegeräten wie z.B. Drucker, Monitore, Beamer oder Scanner. Bei mobilen Endgeräten sind die Peripheriegeräte noch nicht sehr verbreitet, es muss meist ein Umweg über den Desktop-Computer gegangen werden. Dies ist auch ein Grund dafür, dass mobile Endgeräte momentan noch als Ergänzungsgerät angesehen werden. Bei der Softwareentwicklung muss hierfür nach innovative Problemlösungen gesucht werden, z.B. den Einsatz der eingebauten Kamera zum fotografieren von Dokumenten und anschließender Texterkennung oder z.B. die Unterstützung von Apples AirPlay oder anderen Streaming Methoden.

- Eingabemöglichkeiten
  Eine weitere große Herausforderung bei der Entwicklung von mobilen Anwendungen sind die Eingabemöglichkeiten. Bei Desktop-Computern hat man Maus und Tastatur, die sich flächendeckend etabliert haben. Bei mobilen Endgeräten sind Touch-Displays und die damit verbunden Gesten verbreitet. Vereinzelt gibt es noch kleine, physisch eingebaute Tastaturen, Sticks oder Knöpfe. Durch die fehlende Haptik und den kleinen Displays ist die Eingabe bei mobilen Endgeräten meist sehr mühselig und auch nicht so präzise wie z.B. mit einer Maus. Dies stellt die Entwickler vor eine große Herausforderung, sie müssen z.B. die Eingabe auf ein Minimum reduzieren und fehlende Information z.B. nicht vom Nutzer erfragen sondern anderweitig durch Sensoren (Ortsangabe durch GPS) oder Semantik (Adressen aus Adressbuch) beschaffen. Weitere innovative Möglichkeiten sind z.B. Spracheingabe, Gesten über Kamera oder andere Sensoren.

- Prozessorleistung
  Die Prozessorleistung von mobilen Endgeräten ist nicht so hoch wie die aktueller Desktop-Computer, aus diesem Grund ist es schwierig rechenintensive Aufgaben z.B. auf dem Smartphone auszuführen. Deshalb ist es wichtig bei der Entwicklung der Anwendung den Funktionsumfang abzugrenzen und etwaige rechenintensive Aufgaben auszulagern. Jedoch werden die mobilen Prozessoren immer besser und schneller und übertreffen jetzt schon ältere Desktop-Computer.

- Speicherplatz
  Der Speicherplatz bei den mobilen Endgeräten ist meist sehr begrenzt und wie bereits erwähnt, dass die Datenübertragung und die Erweiterbarkeit

beachtet werden muss. Es sollten keine großen Datenmengen lokal gespeichert werden, eine Lösung bieten hier Cloud-Dienste, in welchen z.B. Kundendaten gespeichert sind, die mobilen Endgeräte können dann auf diese einfach zugreifen. Ein wichtiger Punkt hierbei ist die Datensicherheit und der Datenschutz. Entwickler sollten also solche Cloud-Dienste in ihre Anwendungen einbauen, falls auf unternehmensinterne Daten zugegriffen werden soll, und sie sollten dabei auf Sicherheitsmechanismen großen Wert legen.

- Energieverbrauch
  Dies ist ein spezielles Problem der mobilen Endgeräte, zwar wird auch bei Desktop-Computern auf Energiesparsamkeit geachtet, durch die permanente Stromversorgung ist dies jedoch kein Problem. Mobile Endgeräte besitzen einen Akku mit begrenzter Stromversorgung für das Gerät, da das Gerät unterwegs genutzt werden soll und es nicht überall Stromquellen gibt, sollte es den Entwicklern ein wichtiges Anliegen sein, die Anwendung so stromsparend und effizient wie möglich zu gestalten. Hierbei sind umfangreiche Labor- aber auch Feldtests unumgänglich. Außerdem sollten Auto-Save Funktionalitäten integriert sein, damit die Daten nicht verloren gehen[28].

- Betriebssystem und Version
  Wie in den Grundlagen erwähnt gibt es eine Vielzahl an unterschiedlichen mobilen Betriebssystem die mit dem größten Marktanteil sind iOS von Apple, Android von goolge und WindowsPhone von Microsoft. Jedes dieser Systeme hat unterschiedliche Entwicklungsumgebungen und diese haben verschiedene Anforderungen an die Anwendungen. Ein weitere Herausforderung ist die große Fragmentierung, welche in den Grundlagen schon diskutiert wurde. Die Entwickler müssen mit den schnellen Veröffentlichungszyklus der Versionen Schritt halten. Des Weiteren stellt sich die Frage, ob nur für bestimmte Systeme entwickelt werden soll oder alle abgedeckt werden sollen. Dies bedeutet oft für jedes System eine eigene Anwendung zu entwickeln, was mit sehr hohen Entwicklungskosten verbunden ist. Dies lässt sich z.B. mit Plattformübergreifenden Entwicklungstools lösen oder durch WebApps, wie in den Grundlagen schon erwähnt. Diese Heterogenität der mobilen Plattformen stellt die Entwickler vor eine große Herausforderung. Es existieren momentan viele unterschiedliche Betriebssysteme parallel auf dem Markt und es wird sich mittelfristig keines dieser durchsetzen. Dies kann in der Entwicklung zu langen Entwicklungszeiten führen. Aus diesem Grund sollte diese Frage zu Beginn des Entwicklungsprozesses gründlich diskutiert und abgewägt werden[28].

- Auflösung
  Die Auflösung und die Qualität der Displays von mobilen Endgeräten ist sehr gut, die Auflösung lässt sich aber nicht ändern und die unterschiedlichen Geräte haben verschiedene Auflösungen, somit muss bei der Entwicklung auch darauf geachtete werden[10].

- User-Interface
  Ein weiterer wichtiger Punkt bei der mobilen Softwareentwicklung ist das User-Interface, diese muss auf die beschränkte Größe der mobilen Endgeräte und an die Eingabegesten der Multitouch Displays angepasst werden. Dabei funktioniert das bekannte Fenster und Ordner System der Desktop-Computer nicht mehr, es braucht große Bedienfelder. Das User-Interface und dessen Design ist bei dem Entwicklungsprozess sehr wichtig und sollte gesondert berücksichtigt werden[28].

- Sensoren
  Die unterschiedlichen Sensoren wie z.B. der Accelerometer, Gyroskop, Näherungssensor, Umgebungslichtsensor oder der digitaler Kompass können die Anwendung aufwerten und die Bedienung vereinfachen. Es können weitere Funktionen und Dienste angeboten werden wie z.B. NFC zum bezahlen oder Fingerprint-Scanner zur Identifikation und Verbesserung der Sicherheit. Die Entwickler sollten sich überlegen, ob sie diese in der Anwendung verwenden können[28].

- Internetverbindung (3G, GPRS, EDGE, HSPA)
  Die permanente Internetverbindung ermöglicht den ständigen Kontakt mit den Mitarbeitern und verbessert die Kommunikation im Unternehmen. Es lassen sich Internetdienste und Cloud-Services verwenden. Jedoch ist die Internetverbindung meist langsam oder unterbrochen und dies beeinflusst die Leistung der Anwendung[10].

Einige dieser Probleme werden in den Guidelines der jeweiligen Systemanbieter behandelt und Lösungen oder Programmierrichtlinien vorgeschrieben, welche den Softwareentwickler bei der Entwicklung helfen sollen[17][18][21].

Umwelt Besonderheiten

- Kurze Auslieferungszeiten
  Besonders im B2B-Bereich und in der mobilen Softwareentwicklung werden die Auslieferungszyklen immer kürzer. Von der Idee bis zur Einführung vergehen meist nur wenige Monate. Die Entwickler müssen auf diesen Trend geeignet und flexibel reagieren, aus diesem Grund ist ein gutes Entwicklungsmodell bei der Entwicklung unerlässlich.

- Outsourcing
  Bei der Entwicklung ist zu überlegen, ob die Anwendung intern oder extern entwickelt werden soll. Bei B2B-Anwendungen hat man meist eine eigene Entwicklungsabteilung, welche die Anwendungen entwickelt. Bei mobilen Anwendungen wird dies meist ausgelagert oder zumindest ein Teil. Der Vorteil dabei ist, dass Agenturen oder Dienstleister mit Know-how im mobilen Bereich meist sehr schöne und einfache Anwendungen für den speziellen Zweck

entwickeln können. Falls jedoch auch sehr viele interne Schnittstellen genutzt werden sollen bieten sich interne Entwickler an, die freie Kapazitäten haben. Man kann auch die eigenen Entwickler in dem mobilen Bereich Schulen oder Coachen lassen durch einen Dienstleister. Es gibt auch die Möglichkeit z.B. die Android Version selbst zu Entwickeln und die iOS Version durch einen Dienstleister. Es kommt ganz auf den Aufgabenzweck und Ressourcen an[15].

- Skalierbarkeit
  Die individuellen und spezialisierten mobilen B2B-Anwendungen lassen sich relativ leicht für eine überschaubare Anzahl an Kunden ausrollen und leicht auf den aktuellen Stand halten. Sollte es jedoch eine Vielzahl unterschiedlicher Varianten geben, so stellt sich die Frage der Skalierbarkeit. Wie können die vielen unterschiedlichen Gerätetypen mit unterschiedlichen Betriebssystemen und Betriebssystemversionen versorgt und unterstützt werden. Dies erfordert ein gutes Geräte- und Versionsmanagement. Die Entwickler müssen hierfür geeignete Lösungen finden[27].

- Sicherheit
  Sicherheit spielt nicht nur bei der Entwicklung von Desktop-Computern eine wichtige Rolle sondern besonders auch bei mobilen Endgeräten. Hier stellt sich z.B. das Problem, dass mobile Endgeräte von der gleichen Person in unterschiedlichen Rollen genutzt wird (Bring-your-own-device). Es ist also wichtig die privaten und geschäftlichen Daten zu Trennen oder unterschiedliche Sicherheitsstufen zu implementieren[27].

- Integration
  Die Integration in die Geschäftsabläufe und in die vorhandenen IT-Landschaft des Unternehmens ist auch eine große Herausforderung für die Entwickler. Sie müssen die Anwendungen an die bestehende Geschäftsanwendung des Unternehmens anbinden. Das kann bei der Vielzahl von Schnittstellen und technischen Standards zu erheblichen Schwierigkeiten führen. Des Weiteren sind die meisten Prozesse im Unternehmen nicht für die mobile Nutzung ausgelegt und müssen erst mühsam angepasst werden.

- Hoher Grad an Wettbewerb
  Es herrscht ein hoher Grad an Wettbewerb, dies betrifft besonders der B2C-Bereich aber auch immer mehr den B2B-Bereich. Die Entwicklung von mobilen Anwendungen ist sehr einfach geworden, mit Hilfe der Entwicklertools der jeweiligen Anbieter. Es ist jedem möglich der einen Entwickler-Account besitzt eine Anwendung zu erstellen und diese auch in den jeweiligen App Stores zu vermarkten. Die Entwicklung ist so einfach, dass es keine großen Entwicklerteams benötigt und auch die Vermarktung wurde vereinfacht. Die Anwendungen werden in den App Stores ausgestellt, es müssen somit keine großen Marketing Maßnahmen ergriffen werden und die Entwicklerkosten sind sehr gering was für "Hobby-Programmiererßehr attraktiv ist.

- Steigender Mobilitätsbedarf
  Die Unternehmen werden immer flexibler um am Markt bestehen zu können aus diesem Grund braucht man mobile Mitarbeiter und Anwendungen welche diese unterwegs unterstützen. Des Weiteren erfordert die Einfuhr der neuen mobilen Endgeräte aus dem Privatkundenmarkt einen neuen Umgang mit mobiler IT im Unternehmen[24].

- Hohe Dynamik am Markt
  Die Leistungsfähigkeit und die Funktionalität der mobilen Endgeräte steigt immer weiter, während der Preis immer weiter fällt. Auch die Qualität und die Zuverlässigkeit der Netzwerke wächst weiter und wird immer kostengünstiger, es ergeben sich dadurch auch sehr gute und innovative Cloud-Lösungen, welche neue Chancen eröffnen. All dies sind Treiber die für mobile Endgeräte sprechen und auf diese Trends muss bei der Entwicklung eingegangen werden[24].

- Ermittlung der Anforderungen
  Eine große Aufgabe bei der Entwicklung ist die Ermittlung der Anforderungen, es ist sehr schwierig die Bedürfnisse der Kunden genau zu ermitteln. Des Weiteren ändern sich Kundenbedürfnisse sehr schnell, was heute noch aktuell ist, ist morgen schon wieder veraltet. Die Updatezyklen der Betriebssysteme sind sehr kurz und die Entwickler müssen sich darauf einstellen und die neusten Änderungen in die Anwendung integrieren.

- App Stores
  Die App Stores bieten viele Vorteile, besonders für den B2C-Bereich aber auch viele Nachteile. Großer Vorteil dabei ist, dass die Anwendungen an einem zentralen Punkt vermarktet werden, die Kosten dafür übernehmen die jeweiligen App Store Betreiber, jedoch müssen die Entwickler meist 30% des Gewinns abgeben. Die Kunden können so relativ leicht die Anwendungen über die App Stores suchen und herunterladen. Jedoch sind diese Stores für den B2B-Bereich eher ungünstig, da meist die Anwendungen für mehrere Geräte eingekauft werden müssen oder die Anwendungen nur für eine bestimmte Zielgruppe gedacht ist.

- Rechtliches
  Bei dem mobilen Anwendungen gibt es unterschiedliche Vertragspartner zum einen der Entwickler/Anbieter, der Nutzer und der Store-Anbieter. Bei dem Vertragsverhältnis zwischen Store-Betreiber und App-Anbieter gilt meist US-amerikanisches Recht (z.B. im Falle von Apple und Google), die App-Anbieter sind zur Einhaltung der jeweiligen, gesetzlicher Vorschriften verpflichtet. Bei dem Vertragsverhältnis zwischen Store-Betreiber und Nutzer gilt auch meist das US-amerikanische Recht, jedoch gibt es ein Mindestschutz für Verbraucher durch das deutsche AGB-Recht. Der Store-Betreiber muss dabei kein Widerrufsrecht einräumen. Das wichtigste Vertragsverhältnis ist zwischen App-Anbieter und Nutzer, dabei werden die als wie Internet-

Anwendungen behandelt, es gibt eine Impressums-Pflicht, Datenschutz, Preisangabenverordnung, Fernabsatzrecht, sowie Branchenspezifische Gesetze und Verordnungen[15].

- Preismodell
  Die meisten Anwendungen sind relativ klein, besitzen also nur einen beschränkten Funktionsumfang und sind meist nur für einfache kleinere Aufgaben gedacht. Es gibt sehr wenige Anwendungen, die einen kompletten Geschäftsprozess mit all seinen Facetten abdecken. Dies spiegelt sich auch in den Preisen der Anwendungen wieder. Die Zahlungsbereitschaft der Kunden für Anwendungen sind sehr gering, dies bedeutet eine geringe Gewinnspanne für die. Jedoch gibt es auch die sogenannten In-App Einkäufe, diese ermöglichen es z.b. weitere Funktionen frei zuschalten und dafür zu bezahlen.

5 Vergleich

Tabelle 3. bietet eine schnelle Übersicht über die Unterschiede und Herausforderungen zwischen der Entwicklung klassischer Software und der Herstellung mobiler Anwendungen. Diese werden in die einzelnen Phasen des Grundmodells der Softwareentwicklung, sowie in wichtige Bereiche bei der Herstellung mobiler Anwendungen aufgeteilt.
Die Ideen-Phase der beiden Bereiche unterscheidet sich im Prinzip nicht sehr, jedoch liegt der Schwerpunkt der klassischen Entwicklung im stationären Bereich und bei der mobilen Entwicklung auf Mobilität. Dementsprechend sind Zielumgebung, Funktionen und Einsatzzweck zu berücksichtigen. Bei der Auftragsklärungsp-Phase, geht es um die Machbarkeit und die Wirtschaftlichkeit, hier hat es der mobile Sektor leichter, da der Funktionsumfang meist geringer und die Entwicklungskosten überschaubarer sind. Das Risiko ist somit im mobilen Bereich gering. Bei der Konzeptions- und Design-Phase unterscheiden sich die beiden Bereiche sehr. Hier müssen komplett unterschiedliche Ansätze entwickelt werden, was User-Interface und Bedienung angeht. Auch die Zielsysteme sind sehr unterschiedlich und haben ihre spezielle Herausforderungen. Bei der Realisierung unterschieden sich die Bereiche geringfügig in einzelnen Punkten. Meist erweist sich die Integration der mobilen Anwendung in das vorhandene System als schwieriger. Im Marketing- und Distributions-Bereich gibt es ebenfalls kleine unterschiede. Bei der mobilen Softwareentwicklung sind z.B. andere Preismodelle interessant, des Weiteren gibt es Vor- und Nachteile bei den App Stores im mobilen Bereich. Große unterschiede gibt es natürlich im Hardware-Bereich, da es komplett unterschiedliche Ansätze sind, die sich auch in verschiedene Richtungen entwickeln.

6 Zusammenfassung und Ausblick

Aktuell beschränkt sich die Nutzung von mobilen Endgeräten im Allgemeinen noch auf die Kommunikation und die Interaktion zwischen den Mitarbeitern.

Tabelle 3. Übersicht über die Unterschiede und Herausforderungen zwischen klassischer Softwareentwicklung und mobiler Softwareentwicklung

Softwareentwicklu	**klassisch**	**mobil**
Idee		
Zielumgebung	stationäre Desktop-Computer	mobile Endgeräte
Einsatzzweck	Datenverwaltung/bearbeitung	mobiler Datenabruf/mobile Datenbearbeitung, Kommunikation
Auftragsklärung		
Wirtschaftlichkeit	schwierig	einfach
Machbarkeit	mittel, schwierig	einfach
Vergabe	intern, extern	intern, meist extern
Produktportfolio	Eigenständige Produkte, Erweiterungen	mobile Funktionen, Erweiterungen
Konzeption		
User-Interface	klassische Fenster und Ordner Struktur	innovative, neue Gestaltungs- und Strukturmöglichkeiten
Bedienung	Maus und Tastatur	Gesten
Auslieferungszeiten	mittel	kurz
Datenschutz	wichtig	sehr wichtig
Design		
Art	nativ	webApp, nativ, Hybrid, Cross oder Beschreibungssprache
System	Windows, MacOS	iOS, Android, WindowsPhone, BlackBerry, Symbian
Diversität	gering	sehr hoch
Features	viele	kein Feature-Overkill
Realisierung		
Entwicklungszyklen	mittel bis lang	kurz
Teamgröße	mittel bis groß	klein bis mittel
Integration	leicht	schwierig
Skalierbarkeit	leicht	leicht, mittel
Testarten	Labortests	Labor- und Feldtests
Programmgröße	>10000 lines of code	<10000 lines of code
Marketing		
Wettbewerb	hoch	hoch
Werbekosten	mittel	hoch
Distribution		
Verkauf	Datenträger, Download	App Stores, Download
Kosten	Bereitstellungskosten	70/30 Modell der App Stores
Preismodell	Shareware, Freeware	Kostenfrei, Kostenpflichtig, Freemium und In-App-Verkäufe, Abonnement-Modell
Hardware		
Größe/Gewicht	groß, schwer	klein, leicht
Bildschirmgröße	groß	mittel, klein
Erweiterbarkeit	leicht	schwierig
Peripheriegeräte	große Anzahl unterschiedlicher Geräte	wenige
Leistung	sehr hoch	mittel
Speicherplatz	groß und Erweiterbar	gering, meist nicht erweiterbar
Energieverbrauch	hoch	gering
Sensoren	wenige	viele
Datenübertragung	Datenträger, Netzwerk (Intranet, Internet), W-LAN	EDGE, 3G, HSPA, W-LAN, Blutooth

Dies geschieht durch Sprache (Telefonie), Text (E-Mail, SMS, Instant Messaging) oder Video (Video-Telefonie). Jedoch bieten die neuen mobilen Anwendungen die Chance auch abseits dieses Bereiches die Unternehmen zu mobilisieren. Es können die Arbeitsabläufe im Unternehmen durch mobile Anwendungen bereichert und vereinfacht werden. Es ist dem Mitarbeiter z.B. möglich die Reisezeit unterwegs produktiv zu nutzen, durch den Zugriff auf die Unternehmensdaten unterwegs.
Jedoch birgt die Entwicklung mobiler Anwendungen auch neue Herausforderungen im Vergleich zur klassischen Softwareentwicklung. Zu den klassischen Herausforderungen der Softwareentwicklung, die es immer gibt, wie Sicherheit, Leistungsfähigkeit oder Speicherbegrenzungen gibt es eine Vielzahl spezieller Aufgabenstellungen bei der Herstellung mobiler Anwendungen, auf welche die Entwickler im Softwareentwicklungsprozess speziell eingehen müssen. Einige dieser Herausforderungen beziehen sich auf die Größe des mobilen Endgerätes und besonders auf die Mobilität und die daraus entstehenden Probleme wie die Akkulaufzeit. Um auf diese Schwierigkeiten bei der Entwicklung geeignet reagieren zu können muss der Entwicklungsprozess angepasst werden. Es gibt sehr viele unterschiedliche Entwicklungsmodelle, bei der mobilen Softwareentwicklung bieten sich besonders agile Prozesse an, um flexibel auf die Bedürfnisse reagieren zu können. Aber man kann nicht pauschal sagen welches der Entwicklungsmodelle am besten geeignet ist, es kommt dabei immer auf den jeweiligen Einzelfall darauf an und man muss dabei ein geeignetes Gleichgewicht finden. Ein besonderer Schwerpunkt liegt auch auf dem Testen der jeweiligen Anwendung. Die gewonnene Mobilität der Geräte fordert jedoch von den Entwicklern, dass diese nicht nur in Labortest getestet werden sondern auch in Feldtests unter realen Bedingungen. Des Weiteren sollte die Anwendung auf den unterschiedlichen Geräten und unter den unterschiedlichen Systemen getestet werden. Hierbei bieten sich besonders im Unternehmensbereich die Bring-your-own-Device Methode an. Die meisten Mitarbeiter besitzen ein eigenes Smartphone oder Tablet-Computer für den privaten Bereich. Diese können auch einfach für das Unternehmen genutzt werden, der Vorteil davon ist das sich der Nutzer mit dem Gerät vertraut ist. Jedoch gibt es hierbei auch Sicherheitsprobleme, es müssen die Privatdaten von den Unternehmensdaten getrennt werden.
Eines der größten Herausforderungen bleibt jedoch der Umgang mit der Fragmentierung im mobilen Bereich, mittelfristig wird sich keiner der vielen Anbieter direkt durchsetzten. Es bleibt eine Koexistenz der unterschiedlichen Hersteller. Jedoch bleiben Apples iOS und Googles Android ein sehr wichtiger Markt. Man versucht jedoch dieses Problem mit plattformübergreifenden Techniken entgegenzuwirken. Eine der wichtigsten Technologien sind hierbei die WebApps auch hier wird sich in Zukunft einiges tun, diese werden immer besser und die Technologien wie HTML5, CSS und JavaScript werden stetig weiterentwickelt. Langfristig wird sich diese Technologie besonders im B2B-Bereich durchsetzen, da immer mehr Funktionen unterstützt werden und die Anwendung dabei sehr flexibel agiert und besonders im Entwicklungsbereich bietet diese Technologie viele Vorteile und auch Kostenersparnisse.

Literatur

1. Abrahamsson, P.: Mobile-d: An agile approach for mobile application development. OOPSLA '04 Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (2004)
2. Adobe: Phonegap 2.8.1 (zuletzt abgerufen am 15 Juni 2013), http://phonegap.com/download/
3. AG, S.: Sap mobile apps (zuletzt abgerufen am 22 Juni 2013), http://ecohub.sap.com/store/mobility
4. appcelerator: Titanium mobile development environment (zuletzt abgerufen am 15 Juni 2013), http://www.appcelerator.com/platform/titanium-platform/
5. Apple: Apples app store (zuletzt abgerufen am 22 Juni 2013), http://www.apple.com/de/pr/library/2013/05/16Apples-App-Store-Marks-Historic-50-Billionth-Download.html
6. Bassi, D.: Enterprise mobile apps (zuletzt abgerufen am 15 Juni 2013), http://www.research2guidance.com/enterprise-mobile-apps-200-000-in-stores-but-only-14-address-core-enterprise-needs/
7. Beck, K.: Manifesto for agile software development (zuletzt abgerufen am 15 Juni 2013), http://www.agilemanifesto.org
8. BITKOM: Mobile anwendungen in der itk-branche (Mai 2011)
9. Bundesverwaltungsamt: V-modell xt bund (zuletzt abgerufen am 15 Juni 2013), http://www.bit.bund.de/BIT/DE/Standards__Methoden/V-Modell_20XT/node.html?__nnn=true
10. Dongsong, Z.: Challenges, methodologies, and issues in the usability testing of mobile applications. International Journal of Human-Computer Interaction (2005)
11. jQuery Foundation: Announcing jquery mobile 1.3.1 (zuletzt abgerufen am 22 Juni 2013), http://jquerymobile.com/blog/2013/04/10/announcing-jquery-mobile-1-3-1/
12. Gartner, I.: Gartner say worldwide pc, tablet and mobile phone combined shipments to reach 2.4 billion units in 2013 (zuletzt abgerufen am 15 Juni 2013), http://www.gartner.com/newsroom/id/2408515
13. GmbH, W.: Appconkit (zuletzt abgerufen am 22 Juni 2013), http://www.weptun.de/appconkit/architektur/
14. Hans, B.P.: Softwareentwicklung kompakt und verständlich. Vieweg+Teubner (2008)
15. Hans-Jörg Stangor, T.K.: Mobile applikationene - ein leitfaden für unternehmen (November 2011)
16. IEEE: Ieee standard glossary of software engineering terminology (1990)
17. Inc., A.: App review guidelines (zuletzt abgerufen am 22 Juni 2013), https://developer.apple.com/appstore/guidelines.html
18. Inc., G.: Android developer site (zuletzt abgerufen am 22 Juni 2013), http://developer.android.com/develop/index.html
19. Inc., G.: Build version codes (zuletzt abgerufen am 22 Juni 2013), http://developer.android.com/reference/android/os/Build.VERSION_CODES.html
20. Maximini, D.: Scrum - Einführung in der Unternehmenspraxis. Springer-Verlag (2013)
21. Microsoft: Windows phone developer site (zuletzt abgerufen am 22 Juni 2013), http://developer.windowsphone.com/en-us
22. PC-Magazin: Apps im business-bereich liegen im trend (Juli 2012)
23. Peter, H.: Agility kompakt - Tipps für erfolgreiche Systementwicklung. Spektrum Akademischer Verlag (2009)
24. Research, B.: Mobile enterprise solutions (2010)

25. Scharff, C.: Scrum to support mobile application development projects in a just-in-time learning context. Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering (2010)
26. Tore Dyba, T.D.: Empirical studies of agile software development: A systematic review (Januar 2008)
27. Verclas, S.: Smart Mobile Apps. Springer-Verlag (2012)
28. Wasserman, A.: Software engineering issues for mobile application development. FoSER 2010 (2010)
29. Xamarin: Create amazing iphone and ipad apps with c# and .net (zuletzt abgerufen am 22 Juni 2013), http://xamarin.com/monotouch

Kategorisierung von mobilen Anwendungen im B2B-Bereich Empirische Auswertung der Marktanteile Seminar Ubiquitous Systems - SS 13

Yangjun Shen and Nadja Sackmann

Karlsruhe Institute of Technology (KIT), Pervasive Computing Systems - TECO
yangjun.shen@student.kit.edu
sackmann@teco.edu

Zusammenfassung Im B2B-Bereich stehen viele Unternehmen vor der Herausforderung mobile Anwendungen einzuführen. Dazu müssen Unternehmen schnell und kostengünstige, geeignete B2B-Apps identifizieren können. Wie auf dem Markt für B2C-Apps wird dafür eine Möglichkeit der Kategorisierung benötigt. Für diesen Zweck steht eine Vielzahl von Kriterien zur Verfügung. Es gilt Vor- und Nachteile dieser Kriterien zu analysieren, um eine fundierte Entscheidungsbasis schaffen zu können, anhand dessen ein Überblick über den Markt für B2B-Apps geschaffen werden kann.

Keywords: B2B, mobile application, mobile business, B2B-Application, App Store, Security, Katalogisierung

1 Einleitung

Mit dem Siegeszug mobiler Endgeräte auf dem Markt der mobilen Telekommunikation, etablierten sich fast zeitgleich mobile Apps für Privatkunden (Business to Consumer). Dank neuer Mobilfunkstandards konnten höhere Übertragungsraten erreicht werden, wodurch neben mobilem „Browsen“ auch das mobile Arbeiten möglich wurde. Bereits existierende, klassische Services wurden dadurch mobilisiert und neue innovative Services angeboten. Auch im Geschäftsumfeld versuchen immer mehr Unternehmen das Potenzial mobiler Anwendungen für sich zu nutzen [8]. Der Hauptvorteil mobiler B2B-Anwendungen (Business to Business) im Vergleich zu klassischen Desktop-Anwendungen ist die höhere Flexibilität durch nahezu unbegrenzte Mobilität. Mitarbeiter können dadurch im Einsatz ortsunabhängig agieren. Besonders im B2E-Umfeld (Business to Employer) werden mobile Technologien zunehmend eingesetzt. Daraus ergeben sich weitere Märkte und neue B2B-Beziehungen. Unternehmen werden in der Lage sein, mobil verfügbare Dienste für andere Unternehmen bereitzustellen. Tendenziell kann davon ausgegangen werden, dass es auf diesem Feld das größte Wachstumspotenzial im Bereich des e-Commerce gibt [12].

Aktuell sind ca. 200.000 B2B-Apps in verschiedenen App Stores öffentlich zugänglich [4]. Es stellt sich daher die Frage, welche Möglichkeiten es gibt diese sinnvoll zu kategorisieren. Die Kategorisierung sollte in erster Hinsicht einen Gesamtüberblick über den Markt der B2B-Apps ermöglichen. Im ersten Abschnitt dieser Arbeit werden deshalb zuerst mögliche Anwendungsszenarien mobiler B2B-Apps besprochen, es werden Unterschiede zwischen B2B und B2C im Kontext mobiler Apps aufgezeigt und Distributionsmöglichkeiten für mobile Apps kurz erläutert. Im zweiten Teil der Arbeit werden zunächst Merkmale besprochen, mittels derer eine Kategorisierung erfolgen könnte. Anschließend werden die ermittelten Merkmale kurz evaluiert. Das Ergebnis ist die Möglichkeit der Kategorisierung nach Porters Wertschöpfungskette. Des Weiteren werden kurz alternative Kategorisierungsansätze gezeigt. Zum Schluss erfolgt eine kurze empirische Marktanalyse, die zeigt wie sich die aktuelle Verteilung exemplarischer B2B-Apps auf die ermittelten Kategorien derzeit gestaltet.

2 Grundlagen

Im folgenden Abschnitt sollen grundlegende Begriffe erläutert werden, die für das Verständnis der Arbeit nötig sind. Zunächst können zwischen zwei verschiedenen Ausprägungen mobiler Anwendungen unterschieden werden.

Native Apps: Native Apps repräsentieren das „klassische“ Entwicklungsmodell, bei dem Applikationen speziell für eine Plattform unter Benutzung einer bestimmten Programmiersprache (Objective-C für iOS, Java für Android) und entsprechender Software Development Kits (SDKs) entwickelt werden. Versionen derselben App für unterschiedliche Plattformen müssen dabei in der Regel von einem Programmierer vollständig neu entwickelt werden. Darüber hinaus unterscheiden sich die jeweils zur Verfügung stehenden Online-Marktplätze, die für die Distribution der Anwendungen notwendig sind. Ein Entwickler hat daher auch die Aufgabe sich mit den entsprechenden Vorgaben und Richtlinien des Plattformbetreibers auseinanderzusetzen. Dies stellt einen nicht zu unterschätzenden Zeitaufwand dar. Ein Unternehmen, das Anwendungen in einem App Store veröffentlichen möchte, muss sich in den meisten Fällen zunächst vom Plattformbetreiber zertifizieren lassen. Außerdem muss jede Anwendung einen Validierungsprozess durchlaufen, bevor diese von Nutzern des App Stores heruntergeladen und installiert werden können [6].

Mobile-Business-,Enterprise,-B2B-App:
Mobile-Business-, Enterprise, bzw. B2B-Apps sind mobile Anwendungen zur professionellen Nutzung im Geschäftsumfeld, die meist in eine existierende IT-Infrastruktur integriert ist. Die Nutzung dieser Apps erfolgt über mobile Endgeräte wie Smartphones oder Tablet-PCs. Die Abgrenzung der B2B-App gegenüber der B2C-App erfolgt anhand des Nachfragertyps. B2B-Apps adressieren ausschließlich Geschäftskunden. B2B-Apps können den Anwender bei der Erledigung von mobilen Business-Tasks unterstützen. Sie zeichnen sich meist

durch einen klaren und limitierten Funktionsumfang aus. In diesem Sinne ist eine Mobile-Business-App nicht eine vom Desktop auf ein mobiles Gerät transferierte Anwendung, sondern eine spezielle, maßgeschneiderte Lösung für mobile Endgeräte, durch die ausgewählte Business-Tasks beziehungsweise Ausschnitte aus Geschäftsprozessen mobilisiert werden. Nicht unter diese Definition fallen sowohl reine mobile E-Mail-Dienste und mobiles Internet und SMS/MMS-Dienste [7].

Der Begriff **App Store** (zusammengesetzt aus Application und Store), wurde durch den Computer Hersteller Apple Inc. geprägt. Bei einem App Store handelt es sich um eine Online- Distributionsplattform für mobile Programme. Nutzer können in diesen Stores Apps suchen, kaufen und herunterladen. Neben dem „original“ App Store von Apple gibt es eine Vielzahl von Stores anderer namenhafter Hersteller wie beispielsweise Google, Samsung oder Microsoft, in denen Apps für entsprechend kompatible Endgeräten vertrieben werden. Des Weiteren ist es privaten und kommerziellen Entwickler meist möglich eigene Apps gegen eine Verwaltungsgebühr auf der jeweiligen Plattform zu vertreiben.

Mobile Websites/browserbasierte Anwendungen:
Im Gegensatz zu nativen Apps sind mobile Websites, auch Web-Apps genannt, die den Browser von mobilen Endgeräten als Basis nutzen. Daten werden über den Browser geladen und versendet. Browserbasierte Anwendungen sind in aller Regel plattformunabhängig. Sie werden mit gängigen Technologiestandards wie HTML5, JavaScript und CSS programmiert. Es findet meistens eine Anpassung hinsichtlich der Display-Größe und Hardware von mobilen Endgeräten statt. Anders als bei Apps ist keine Installation notwendig [6].

Die folgende Arbeit befasst sich hauptsächlich mit der Kategorisierung von nativen Apps als mobile Anwendung im B2B-Bereich, da native Apps über App Stores verschiedener Plattformbetreiber bezogen werden können. Die Betreiber müssen daher Informationen bezüglich verschiedener Leistungsmerkmale veröffentlichen. Diese Informationen können genutzt werden, um daraus sinnvolle und praxisrelevante Ansätze zu Kategorisierung von B2B-Apps abzuleiten.

3 Mobile Anwendungen im B2B-Bereich - Marktanalyse

Mit der Einführung neuer Netzstandards (UMTS) und der damit verbundenen flächendeckenden Verfügbarkeit des mobilen Internet, sowohl für private als auch für kommerzielle Nutzer, fanden zunächst B2C-Apps Einzug in die Welt der mobilen Anwendungen [6]. Zu Beginn beschränkten sich B2C-Apps eher auf kleinere, im Alltag nützliche Aufgaben oder Spiele. Diese bildeten den Grundstein für Apps, die sich bspw. der Organisation von Terminen widmeten und somit bereits im B2B-Bereich genutzt werden konnten. Der Begriff B2B im Zusammenhang mit mobilen Anwendungen lässt sich demnach nicht klar abgrenzen. Dennoch wurde das Potenzial mobiler Anwendungen bereits 2001 als sehr hoch eingeschätzt [20].

3.1 B2B versus B2C

Der Begriff Business-to-Business (B2B) betrachtet im Allgemeinen Beziehungen und Transaktionen zwischen Unternehmen. Diese Definition ist jedoch unzureichend um das B2B-Umfeld vor dem Hintergrund mobiler Anwendungen zu beschreiben. Sülze charakterisiert B2B folgendermaßen:

> The understanding of B2B is such that it does not only describe external communication and transaction functions, but also relates to the flow of information within the company, i.e. between employees, departments, subsidiaries and branches.

Die Definition unterscheidet sich insofern von denen anderer Autoren, da Sülze neben den zwischenbetrieblichen Aspekten des B2B auch die innerbetrieblichen Vorgänge mit in den Fokus rückt. Vor dem Hintergrund des E-Commerce wird der Begriff in der Literatur oft mit B2B-Electronic-Commerce (B2B-E-Commerce) bzw. B2B-Electronic-Business (B2B-E-Business) gleichgesetzt. Im mobilen Bereich gibt es noch die Begriffe Mobile Commerce oder Mobile Business, die in der Regel die Nutzung von mobilen Endgeräten bei der Durchführung von Transaktionen zum Ausdruck bringen sollen. Die folgende Arbeit folgt der Definition von Sülze.

Mit der Einführung mobiler Anwendungen in Unternehmen sind zudem oftmals größere, organisatorische Veränderungen und Geschäftsprozessanpassungen verbunden. Bei Themengebieten wie dem Supply Chain Management sind außerdem Lieferanten und Kunden einzubinden und entsprechende Schnittstellen zu schaffen. Dies begründet ebenfalls die langsamere Entwicklung von mobilen Anwendungen im Geschäftskundenbereich [15].

Die Durchdringung von mobilen Technologien im B2B-Bereich ist im Vergleich zum B2C-Bereich gering [9]. Dies hat vielfältige Gründe. Vertriebsorientierte Anwendungen im B2B-Bereich stehen bspw. vor der großen Herausforderung, eine hohe Anzahl verschiedener Geschäftsbedingungen, Preise, Produktkataloge und Werbematerialien zu verwalten. Hinzu kommt eine Fülle von internen, firmenspezifischen Vorgaben in Form von Vertragsbestimmungen, Einkaufsvorschriften und Reportingstandards, die es in einem zum Teil komplexen Geflecht aus Zulieferern, Distributoren und Partnern zu berücksichtigen gilt. Vor ähnlichen Herausforderungen stehen prinzipiell auch andere Unternehmensteilbereiche (Marketing & Vertrieb, Kundensupport, Accounting usw.) [8].

Weiter gilt es zu beachten, dass B2B-Kunden auch gleichzeitig B2C-Kunden sind, und somit die gleichen Qualitätsstandards bzgl. Funktionalität und Aktualität der mobilen Anwendungen erwarten, wie sie auf dem B2C-App-Markt bereits zum Standard zählt. Auf Vertriebsebene bedeutet dies bspw. konsistente Angaben von produktbezogenen Informationen (Preis, Verfügbarkeit usw.) auf allen bereitgestellten Vertriebskanälen des Produktanbieters. Die einfache Lösung, nämlich die Adaption von B2C-Apps auf den B2B-Bereich scheitert bereits an der weiter oben beschriebenen Komplexität des B2B-Umfelds. Dies erklärt auch, warum viele Firmen noch auf klassische Geschäftsportale und nicht auf moderne mobile Technologien setzen [4].

Research-2guidance, ein Spezialist für Beratungs- und Marktforschungsunternehmen schätzt zwar, dass es derzeit auf dem B2B-Markt ca. 200.0000 Anwendungen in den verschiedenen App Stores gibt. Bisher würden dennoch lediglich nur 14 Prozent der verfügbaren Apps wichtige Unternehmensfunktionen adressieren. Dies sei aber auch darauf zurückzuführen, dass sich der Markt erst in der Entwicklungsphase befindet. Ein weiteres Problem stellt die Identifizierung und Suche von geeigneten B2B-Apps dar. Dies sei häufig mit hohem Zeitaufwand und signifikanten Kosten verbunden [26].

Tabelle 1 zeigt die wesentlichen Unterschiede zwischen B2B und B2C auf. Es ist zu erkennen, dass beide Märkte sich in nahezu allen relevanten Punkten unterscheiden. Vor allem die hohe Produktkomplexität, die B2B-Beziehungen auszeichnen, erschwert die Verbreitung von B2B-Apps bisher. Die meist geringe Abnehmerzahl im Vergleich zum B2C-Markt für Apps und die hohe Individualisierung der B2B-Apps stellen weitere signifikante Unterschiede dar.

Tabelle 1. Unterschiede B2C vs. B2B in Anlehnung an Christmann u. Hagenhoff [9].

	B2C	**B2B**
Abnehmerzahl	hoch	gering
Entscheidungsart	eher emotional	rational, ökonomisch
Entscheidungsstruktur	individuell	formalisierte Gruppenentscheidung
Individualisierung	idR. keine	hoch
Kundenbeziehung	anonym, instabil	langfristig, ggf. Kooperation
Markttransparenz	gering	eher hoch
Nachfrageart	direkt	indirekt
Persönlicher Bezug zum Produkt	ja	eher nein
Preisbildung	marktorientiert	kostenorientiert
Produktkomplexität	niedrig	eher hoch
Wettbewerbsdruck zw. Nachfragern	nein	ja
Ziel	Verbrauch, Verwendung	Verarbeitung, Produktion

3.2 Einsatzpotenzial von B2B-Apps

Die Anwendungsbereiche von B2B-Apps sind grundsätzlich sehr ähnlich mit jenen klassischer Unternehmens-Software. Neben Customer Relationship Management (CRM), Supply Chain Management (SCM), Enterprise Resource Planing (ERP), Business Intelligence (BI) und diversen Kommunikationslösungen finden sich auch Office-Programme und Software für das Dokumentenmanagement wieder. Bisher werden B2B-Apps aber hauptsächlich im Bereich der Verbesserung von Geschäftsprozessen, zur Unterstützung des (CRM) und im Bereich des Sup-

ply Chain Management SCM eingesetzt [8]. B2B-Anwendungen können sowohl unternehmensintern oder auch zwischen Unternehmen eingesetzt werden.

In allen Branchen können B2B-Apps intern eingesetzt werden. Im Bereich Business Intelligence können Reportings und Dashboards auch mobil zur Verfügung gestellt werden. Das Supply Chain Management kann Apps nutzen um Monitoring zu betreiben. Selbst im Security Bereich können Apps helfen, beispielsweise die Arbeitssicherheit in Produktionshallen zu erhöhen. Führungskräfte könnten dadurch auch aus der Ferne Routinearbeiten, wie z.B. Bearbeitung von Genehmigungsanträge, zuverlässig verrichten, für die sie normalerweise persönlich hätten anwesend sein müssen. Unternehmensintern werden B2B-Apps eingesetzt um Organisationsprozesse in den jeweiligen Fachabteilungen im besten Fall zu automatisieren oder zumindest zu beschleunigen. Häufig genanntes Beispiel ist die Abrechnung von Reisekosten durch Mitarbeiter. Außerdem sind sie für die mobile Überwachung und Steuerung von technischen Anlagen besonders geeignet. Unternehmensinterne mobile Lösungen sind in der Regel leichter zu etablieren und zu kommunizieren, während Geschäftsbeziehungen zwischen Unternehmen bereits an rechtlichen Hürden scheitern können [5].

Im produzierenden und dienstleistenden Gewerbe können B2B-Apps die Kommunikation mit dem Kunden im Service und Support erleichtern. Die App bietet dem Kunden in Kombination mit Social Media Ansätzen einen zusätzlichen Kommunikationskanal. Als Beispiel kann die Bearbeitung von Reparatur- und Supportanfragen genannt werden, die mittels App mobil verwaltet werden können. In der Dienstleistungsbranche können Apps auch zu Monitoring und Steuerungszwecken eingesetzt werden. Bereits jetzt setzen viele Unternehmen mobile Endgeräte ein, um die eigenen Vertriebs- und Servicemitarbeiter zu unterstützen. Dies erfolgt in Form der mobilen Bereitstellung von Informationen aus dem firmeneigenen CRM-System, mit dem Ziel die Beratungs- und Servicequalität beim Kunden zu optimieren.

Die steigende Nachfrage nach mobilen Anwendungen auf dem B2B-Markt führt zu der Frage, aus welchen konkreten Gründen Unternehmen bereit sind mobile Apps zu entwickeln. In einer 2011 durchgeführten Online-Umfrage der BITKOM (Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.) in der ITK-Branche (Informations- und Kommunikationstechnologie) mit rund 518Teilnehmer gaben 73% die Erweiterung des Produktportfolios und jeweils 57% verbesserte Kundenbindung bzw. verbesserter Kundenservice als Hauptmotivation an, in den Markt für mobile Anwendungen zu investieren. 53% der entwickelnden Unternehmen gaben an, durch Mobile Apps Neukunden gewinnen zu wollen, dagegen glauben nur 25% an eine Verringerung der Kosten z.b. in den jeweiligen Fachabteilungen [8].

In der Rubrik welche mobile Technologie die nächsten 5 Jahre im B2B-Markt dominieren wird, gaben 61% der Befragten browserbasierte Anwendungen, die für mobile Endgeräte optimiert wurden, an. Lediglich 31% sprachen sich für Apps aus. Im B2C Markt hingegen stimmten 72% für Apps und nur 22% für browserbasierte Anwendungen an. Trotz der Umfrageergebnisse ist ein stetiges Wachstum auch auf dem B2B-App-Markt zu verzeichnen [8].

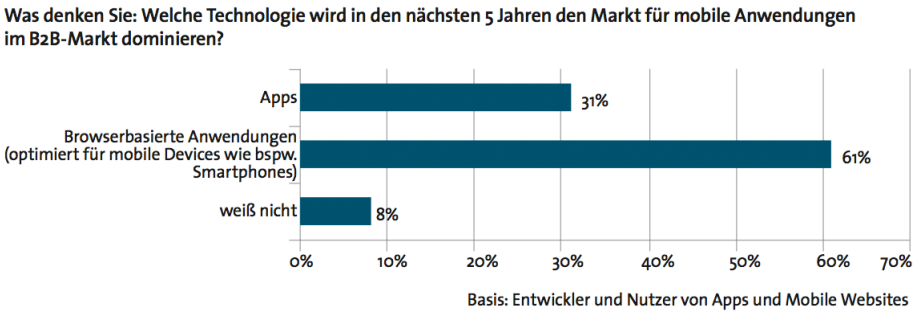


Abbildung 1. Umfrageergebnis - Technologie mobiler Anwendungen B2B-Bereich [8].

In einer weiteren Umfrage von Anfang 2012 gaben bereits 76% bzw. 66% der befragten Entwickler an künftig für Android bzw. iOS entwickeln zu wollen. Nur 53% gaben an Mobile Websites entwickeln zu wollen [27]. Grundsätzlich sollten sich Unternehmen bewusst und strategisch entscheiden, ob und welche mobile Apps zum Einsatz kommen sollen. Vor jeder Entwicklung bzw. Einführung von B2B-Apps muss eine Reihe allgemeiner Überlegungen im Unternehmen stehen. Obwohl mobile Apps ein hohes Potenzial von Mehrnutzen mit sich bringen können, bringen sie auch eine Änderung der Arbeitsläufe, die im Unternehmen klar kommuniziert werden müssen, damit die Akzeptanz unter den Mitarbeiter vorhanden ist.

3.3 Distribution mobiler Anwendungen und Beurteilung

Mit dem Erfolg des ersten App Stores von Apple für B2C-Kunden, zeichnet sich neuerdings ein Trend in Richtung B2B-Stores ab [25]. Grundsätzlich funktioniert der Vertrieb von B2B-Apps ähnlich wie bei B2C-Apps, über App Stores. Während im B2C-Bereich Einzellizenzen verkauft werden, gibt es im B2B-Bereich ein Bedarf für Volumenlizenzen. Apple erkannte diesen signifikanten Unterschied und startete 2011 einen B2B-App-Store (auch Enterprise App Stores genannt) in den USA, der ein spezielles Volumenlizenzprogramm für Geschäftskunden bereitstellt, das es B2B-Kunden ermöglicht, individuelle App-Lösungen von Drittentwicklern passend zu den jeweiligen Anforderungen zu erwerben. Apps können dadurch für mehrere Mitarbeiter über eine einzelne Rechnung erworben werden. Dabei ist es möglich eine beliebige Stückzahl anzugeben. Nach dem Kauf der Volumenlizenz erhält der Käufer Gutscheine, die er an die Mitarbeiter verteilen kann. Diese können anschließend die benötigte App, wie gewohnt über den App Store herunterladen. Apps, die im B2B-Store angeboten werden, können nicht im „normalen“ B2C-Store gefunden oder heruntergeladen werden. Aktuell bietet Apple ihre B2B-Lösung bereits in neun weiteren Ländern (u.a. Deutschland) an [1].

Anders als Apple bietet Google bisher keine eigenständige Lösung für Geschäftskunden an. Google Play Store stellt aktuell nur die Möglichkeit bereit,

über den sogenannten „Google Play Private Channel“ den Nutzerkreis einzuschränken, die Zugang zu den Apps erhalten können [17]. Auch die deutsche Telekom bietet mit dem „Telekom Business Marketplace“ eine Plattform zum Vertrieb von dritt-, und eigenentwickelten B2B-Apps an. Andere namhafte Hersteller von mobilen Endgeräten wie Sony, Blackberry, Samsung und LG verfügen zwar über B2C-App Stores, für den B2B-Bereich gibt es bisher keine equivalente Distributions- und Erwerbsmöglichkeit von B2B-Apps für Entwickler bzw. Firmenkunden. Neben Google und Apple betreibt auch SAP einen eigenen App Store, der sich SAP Store nennt. Im Gegensatz zur Konkurrenz, die hauptsächlich Apps von Drittentwickler vertreiben, bietet SAP nur eigene Entwicklungen an.

Während der Markt für App Stores im B2C-Bereich von namhaften Anbietern dominiert wird, gibt es im B2B-Umfeld bereits eine Vielzahl von App Stores, die von kleinen bis mittleren Unternehmen betrieben werden. Laut Gartner, einem auf dem Gebiet für Entwicklungen in der IT spezialisiertes Marktforschungsunternehmen, wird sich der B2B-App Markt zu einem komplexen Geflecht aus einzelnen, hochspezialisierten Firmen entwickeln, die ihr App-Angebot für bspw. bestimmte Geräte oder Branchen limitieren. Unternehmen könnten dadurch gezwungen sehen, auf einheitliche Standards im Bereich des Erwerbsprozesses verzichten zu müssen [25].

Möchte man die Frage nach der adäquatesten Methode B2B-Apps zu verteilen beantworten, müssen verschiedenste Aspekte berücksichtigt und individuell auf die Bedürfnisse des Unternehmens abgestimmt werden. Folgende Auflistung gibt mögliche Kriterien an, die bei der Verteilung von B2B-Apps in Betracht gezogen werden müssen [19].

Sicherheitsanforderungen:

- Sichere Verteilung
- Begrenzung des Nutzerkreises: Kontrolle, wer App herunterladen darf
- Begrenzung des Nutzerkreises: Kontrolle, wer App verwenden darf
- Sicherheit der Daten, die bereitgestellt werden
- Zugriffe auf interne Server des App-Bereitstellers

Verwaltungsanforderungen:

- Vorhandensein eines Verwaltungstools (unabhängig von Zielplattform)
- einfache Benachrichtigung an alle Anwender der App über bereitstehende Updates
- Monitoring: Wer hat App heruntergeladen und wer verwendet sie?

Enterprise Apps, die lediglich für den internen Einsatz hergestellt werden und somit nur von unternehmenseigenen Mitarbeiter verwendet werden, müssen in der Regel nicht den Weg über App Stores gehen. Diese können einfacher in bestehende Mobile Device Management Systeme (MDM) direkt eingespeist werden. Mitarbeiter können diese Apps dennoch auch mobil, über spezielle Distributionskanäle von App Store-Anbieter herunterladen. Die Verteilung von B2B-Apps

über MDM ist zudem die sicherste Möglichkeit für App-Hersteller die Kontrolle über installierte Apps zu behalten. In einer Umfrage von BITKOM glauben sowohl Entwickler als auch Nutzer, dass sicherheitsrelevante Aspekte in den kommenden fünf Jahren für die Weiterentwicklung des Marktes für mobile Anwendungen entscheidend sein werden [7]. Abbildung 2 zeigt das Gesamtergebnis der Umfrage.

Eine Verteilung über öffentliche App Stores ist in der Regel nicht für B2B-Apps zu empfehlen, da es aufgrund mangelnder Zugangskontrolle aus Unternehmenssicht zu ungewünschten Situationen kommen kann. Gerade in Bezug auf sich unterscheidenden, nationalen Datenschutzrichtlinien verschiedener App Store Anbieter, gestaltet sich die Verteilung von B2B-Apps über öffentliche App Stores als schwierig [19].

Eine dritte Möglichkeit ist es einen unternehmensintern App Store zu betreiben. Diese Möglichtkeit folgt ebenfalls dem MAM-System-Ansatz. Dabei wird die Verteilung von B2B-Apps durch ein dediziertes Mobile Application Mananagement System (MAM-System) verwaltet. Dieses System übernimmt die Verteilung und das Monitoring der Apps. Das Löschen von Apps erfolgt über einen externen Service und kann nur durch ein MDM-System erfolgen.

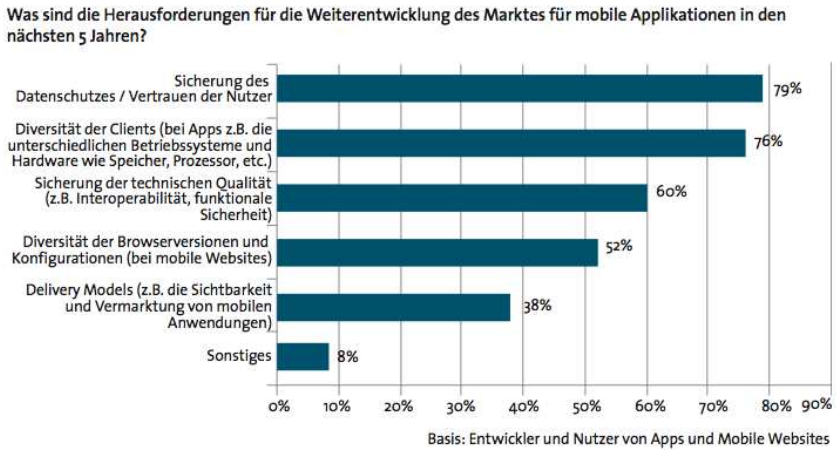


Abbildung 2. Umfrageergebnis - Herausforderungen für den Markt der mobilen Anwendungen [8].

Abbildung 2 zeigt das Ergebnis einer BITKOM-Umfrage, aus der hervor geht, dass sowohl Entwickler als auch Nutzer, die Sicherung des Datenschutzes/Vertrauen der Nutzer als die größte Herausforderung für die Weiterentwicklung mobiler Anwendungen, ansehen.

4 Kategorisierung von Mobilen Anwendungen

Mit dem steigenden App Angebot von B2B-Lösungen stehen Unternehmen vor der Herausforderung nutzenstiftende mobile Anwendungen für den B2B-Einsatz schnell und kostengünstig identifizieren zu können. Hierfür sind intuitive Kategorien hilfreich, die nach bestimmten Kriterien App-Cluster bilden, so dass ein Überblick über den Markt geschaffen werden kann. Der Ansatz der klassischen Literaturrecherche auf dem Gebiet der mobilen Anwendungen gestaltete sich als schwierig, da sich der Markt in einer frühen Entwicklungsphase befindet und die wissenschaftliche Auseinandersetzung mit dem Thema noch am Anfang steht. Um dennoch eine saubere Kategorisierung vornehmen zu können, wurde eine explorative Recherche durchgeführt, um einen Überblick über bereits vorhandene mobile Anwendungen zu erhalten. Anschließend wurde versucht, verschiedene Kategorisierungssichten auf den Markt der mobilen B2B-Anwendungen zu geben.

Im B2C-Bereich werden Apps in der Regel anhand ihres Funktionsumfangs kategorisiert. Google Play Store beispielsweise trifft zuerst eine grundsätzliche Kategorisierung in „Games“ und „Applications“. „Applications“ wiederum in 26 Unterkategorien unterteilt. Diese reichen von „Education“ über „Sports“bis hin zu „Lifestyle“ und „Finance“. Im B2B-Bereich mangelt es derzeit noch an einer übersichtlichen Kategorisierungsmöglichkeit. In den kommenden Kapiteln soll daher eine sinnvolle Kategorisierungsmöglichkeit auch für den B2B-Bereich erarbeitet werden.

4.1 Kategorisierungsmerkmale

Kategorisierung ist die Zusammenstellung von Entitäten nach bestimmten Merkmalen. Diese Merkmale gilt es für B2B-Apps zu identifizieren und zu systematisieren. Dabei stellt sich die Frage welche Merkmale für welchen Zweck als sinnvoll erachtet werden können. Anhand der gewonnenen Erkenntnisse sollen schlussendlich sinnvolle und praxistaugliche Kategorien erstellt werden.

Jede B2B-App verfügt über eine begrenzte Anzahl von öffentlich, abrufbaren Attributen. Diese können in der Regel in jedem öffentlich zugänglichen App Store abgerufen werden. Als Datenbasis für diese Arbeit wurden die zwei umsatzstärksten App Stores gewählt: Apple App Store und Google Play Store [16]. Aus der Recherche in den genannten App Stores können anfänglich, die in Tabelle 3 zusammengefassten Merkmale, als intuitiv relevant angesehen werden.

Als Beispiel listet Tabelle 2 alle Merkmale der App SAP CRM Sales mit der entsprechenden Merkmalsausprägung auf.

Aus Tabelle 2 ergeben sich folgende Merkmale, die potenzielle Merkmale für eine Kategorisierung darstellen.

Folgende Merkmale wurden nicht berücksichtigt: Sprache, letzter Aktualisierungszeitpunkt, Größe, Version und Anzahl Installationen in den letzten 30 Tagen (Information nur in Google Play Store verfügbar). Die Sprache der App spielt in der Regel nur eine geringfügige Rolle, da fast alle Apps Englisch als Standardsprache verwenden und die meisten Mitarbeiter im B2B-Umfeld über

Tabelle 2. SAP CRM Sales - Beispielausprägungen

Merkmal	Ausprägung
Preis	kostenlos
Kundenbewertung	2.6 in Play Store, keine Bewertung in Apple App Store
Plattformunterstützung	iOS und Android
Funktionen	**Beschreibung der Funktionen**
OS-Kompatibilität	ab Android 2.0.3 bzw. ab iOS 4.3
Tablet-Kompatibilität	SmartPhone und Tablet
Zugriffsbefugnisse	Zugriff auf Kamera, Logdaten über Telefongespräche usw.

Tabelle 3. Liste relevanter Kategorisierungsmerkmale

Merkmal	Ausprägung
Preis	Euro / Dollar
Kundenbewertung	1-5 Sterne
Plattformunterstützung	iOS / Android
Funktionen	**Volltext**
OS-Kompatibilität	Angaben über Kompatibilitätsbeschränkungen
Tablet-Kompatibilität	Angaben über Unterstützung von Tablets
Zugriffsbefugnisse	Zugriffsbefugnisse der App

Englischkenntnisse verfügen. Das Merkmal „letzter Aktualisierungszeitpunkt“ist zwar ein Indikator für die Aktualität der App, ist für eine Kategorisierung aber nicht tauglich. Die Größenangabe dient lediglich als Zusatzinformationen, da dieses Kriterium für moderne Smartphones keine Rolle spielt. Die Versionsangabe kann ebenfalls nicht als Kategorie aufgenommen werden, da diese lediglich den internen Stand der Softwareentwicklung widerspiegelt. Die Anzahl der Installationen in den letzten 30 Tagen gibt lediglich einen Trend wider und ist ohne die absolute Anzahl der Installationen nicht aussagekräftig. Außerdem ist dieses Merkmal nur im Google Play Store einsehbar.
Der Umgang mit Datenschutzrichtlinien und andere, die Sicherheit der Daten betreffende Informationen, variiert je nach App Store-Betreiber. Zum Teil bleiben diese Angaben vollständig verborgen, oder können erst nach der Installation der App auf dem mobilen Endgerät eingesehen werden. Der Apple App Store zum Beispiel, gibt nahezu keine Angaben bzgl. Verwendung von gespeicherten Daten preis. Im Google Play Store werden unter „Permissions“ hingegen sicherheitsrelevante Merkmale aufgeführt. Diese Merkmale sind in folgende Kategorien unterteilt:

- **Standort:** Zugriff auf Standort des mobilen Endgeräts (über GPS und WLAN)
- **Netzwerkkommunikation:** Zugriff auf Protokolldaten anderer Apps
- **Telefonanrufe:** Zugriff auf alle Telefonfunktionen des mobilen Endgeräts

- **Speicher:** Änderung und Löschung von USB-Inhalten
- **Ihre Konten:** Abruf von Konteninformationen auf dem mobilen Endgerät, auch jene die von anderen Apps installiert wurden
- **System-Tools:** Steuerung von Bluetooth, Kamera, USB, Virbrationsalarm usw.

Schlussendlich stellt sich die Frage, nach welchen Kriterien eine Katalogisierung von B2B-App sinnvoll ist. Jedes der oben genannten Merkmale stellt eine Möglichkeit dar. Um diese Frage beantworten zu können, muss zunächst geklärt werden, mit welchem Ziel eine Katalogisierung stattfinden soll. Wie im vergangenem Kapitel beschrieben verfügt der Einsatz mobiler Anwendungen über keinen Selbstzweck. Dies bedeutet, dass jeglicher Einsatz der Technologie auch ein konkretes oder weniger konkretes Unternehmensziel adressieren sollte.

4.2 Erstellung von Kategorien

Im vorangegangenen Kapitel wurden B2B-Apps aus unterschiedlichen Unternehmensabteilungen präsentiert und deren Funktionsumfang kurz umrissen. Im folgenden soll eine intuitive Kategorisierung zunächst nach Unternehmensabteilungen und anschließend nach Aufgabentypen durchgeführt werde. Dabei gilt es zu beachten, dass jede Kategorie ein konkretes Unternehmensziel adressiert, dass auch tatsächlich durch den Einsatz von mobilen Anwendungen profitieren kann. Die Verwendung der Wertschöpfungskette nach Porter bietet sich in diesem Fall an, da sämtliche wertschöpfenden Elemente und Tätigkeiten eines Unternehmens beschrieben werden. Je nach Branche muss die Priorität der durchgeführten Aktivitäten entlang der Wertschöpfungskette abgestimmt werden [23]. In Handelsunternehmen liegt das Wertschöpfungspotenzial hauptsächlich in der Logistik, so dass hier Wettbewerbsvorteile geschaffen werden. In Dienstleistungsunternehmen werden Aktivitäten, die im Rahmen des Operations oder in der Produktion durchgeführt werden entscheidend sein. Logistik spielt hingegen eher eine untergeordnete Rolle. Um konkrete Ansätze bzgl. des Einsatzes mobiler Anwendungen in den jeweiligen Teilbereichen durchführen können, müsste also zunächst eine Priorisierung der einzelnen wertschöpfenden Elemente, durchgeführt werden.

Folgende Auflistung zählt die fünf primären Aktivitäten der Wertschöpfungskette nach Porter auf, um diese anschließend kurz zu beschreiben.

1. **Eingangslogistik**
   Eingangslogistik kann als Empfang, Lagerung und Distribution von Betriebsmitteln für das Produkt beschrieben werden. Sie ist außerdem verantwortlich für den Materialfluss und zuständig für die Lagerhaltung und Bestandskontrolle der Betriebsmittel.

2. **Operation/Produktion**
   Tätigkeiten, die sich mit der Umwandlung der Inputs in Outputs beschäftigen. Hauptaufgaben sind maschinelle Bearbeitung, Instandhaltung der Betriebsmittel, Prüfverfahren, Montage, Betrieb von Anlagen usw.

3. **Ausgangslogistik**
   Sammlung, Lagerung und Distribution der Outputs (produzierte Waren) an den Abnehmer. Planung von Termin für Auftragsabwicklung.

4. **Marketing & Vertrieb**
   Bereitstellung von Ressourcen, die dazu verwendet werden, um potenzielle Kunden zum Kauf des hergestellten Produkts zu bewegen.

5. **Kundendienst**
   Installierung bzw. Reparaturen von des Produkts beim Abnehmer. Außerdem zählen Ersatzteillieferungen und Produktanpassung zur Förderung der Werterhaltung zu den Aufgaben des Kundendienstes.

Abbildung 3. Wertschöpfungskette nach Porter [23].

Folgende Auflistung zählt die vier sekundären Aktivitäten der Wertschöpfungskette nach Porter auf und erläutert diese kurz.

1. **Unternehmensinfrastruktur**
   Aktivitäten in der Unternehmensinfrastruktur nehmen Einfluss auf die gesamte Wertschöpfungskette. Typische Aufgaben liegen im Management bzw. in der Unternehmensführung und betreffen Qualitätskontrollen, Rechtsfragen und Finanzen. Aber auch Planung und Kontakte zu Behörden und staatlichen Institutionen gehören hierzu.

2. **Personalwirtschaft**
   Die Personalwirtschaft ist als unterstützende Aktivität anzusehen. Ihr Einfluss erstreckt sich über das gesamte Unternehmen und unterstützt außerdem die gesamte Wertschöpfungskette. Dies wird durch die Tatsache verdeutlicht, dass das Human Ressource Management sowohl bei der Einstellung und Entlassung von Mitarbeitern tätig ist,als auch bei Entscheidungen die die gesamte Wertkette betreffen, wie bei der Teilnahme an Tarifverhandlungen, mitwirkt.
3. **Technologieentwicklung**
   Die Technologieentwicklung beschränkt nicht nur auf Produktverbesserungen, sondern bemüht sich auch Arbeitsabläufe und Verfahrensverbesserungen, vorzunehmen. Da jede andere Wertschöpfungsaktivität auch von der Technologieentwicklung profitieren und unterstützt werden kann, befindet sich dieser Punkt unter den sekundären Aktivitäten. Porters Kritik richtet sich an Unternehmen, die sich lediglich auf Produkttechnologie bzw. Grundverfahren in der Fertigung konzentrieren. Dabei würden Technologien in anderen Wertaktivitäten außer acht gelassen werden. Dies sollte vermieden werden, denn Technologien befinden sich in jeder einzelnen Wertschöpfungsaktivität und es sollte stets darauf geachtet werden, ob es außerhalb der eigenen Branche Technologien gibt, die sich auf das eigene Tätigkeitsfeld anwenden ließen. Porter stellt weiterhin die Beobachtung auf, dass sich Technologiesprünge außerhalb der ursprünglichen Branche am wahrscheinlichsten stattfindet.
4. **Beschaffung**
   Die Unterscheidung zwischen der primären Aktivität Eingangslogistik und der unterstützenden Aktivität Beschaffung ist grundlegend. Demnach stellt Beschaffung nicht die eigentliche Tätigkeit des Einkaufs von Input dar, sondern bezeichnet lediglich die Funktion des Einkaufs. Beschaffungsfunktion bezieht sich somit auf alle Aktivitäten der Wertschöpfungskette und ist damit in allen Bereichen des Unternehmens zu finden.

4.3 B2B-App-Beispiele in der Praxis

Im folgenden sollen diverse Apps vorgestellt werden, die bereits auf dem Markt erhältlich sind, um die Einsatzweise-, und gebiete von Apps im B2B-Umfeld zu dokumentieren. Die Beschreibung der B2B-Apps erfolgt kurz und ohne zu sehr ins Detail zu gehen, da dies für die anschließende Kategorisierung nicht notwendig ist und außerdem den Rahmen dieser Arbeit sprengen würde. Die Einteilung erfolgt zunächst in die fünf primären Aktivitäten der Wertschöpfungskette. Dazu wurden B2B-Apps, die mit „Business“ (Google Play Store) bzw. „Wirtschaft“ (Apple App Store) getaggt wurden, beispielhaft ausgewählt. Anschließend wurde anhand des Merkmals Funktionsumfang manuell versucht die Hauptfunktionen der App zu identifizieren, um eine anschließende Einteilung in eines der primären Aktivitäten nach Porter zu ermöglichen.

- **Eingangslogistik**
  Aventeon Logistics.ONE Aventeon Logistics.ONE unterstützt Speditionen

bei ihrer Arbeit, indem die IT-Unterstützung auch auf die mobil ablaufenden Geschäftsprozessteile ausgedehnt wird und jederzeit eine Verbindung zwischen Fahrer, Auftraggeber, Planer und IT-System besteht. Dabei können auch – wie es branchentypisch ist – Subunternehmer mit eingebunden werden. Logistics.ONE wird beispielsweise durch die Meeus Gruppe, Netko oder Lavans eingesetzt [3].
Data One Mobile Warehouse Management Das „Data One Mobile Warehousemanagement“ gehört zu den Data One Mobile Solutions und ermöglicht die mobile Durchführung lagerspezifischer Geschäftsprozesse. Dazu stehen folgende Funktionen im WM-System zur Verfügung: Wareneingang, Inventur, Warenausgang und Kommisionierung und Warehouse Cockpit [22].

- **Operation & Produktion**
  cobra Mobile CRM Die Software cobra Mobile CRM von CRMPioniers bietet beispielsweise neben den herkömmlichen Kontakt- und Adressmanagementfunktionen, ein für HD- Displays optimierte Oberfläche an, die es ermöglicht auch auf einem mobilen Endgerät deutlich mehr Informationen darzustellen als auf einem nicht HD-Display [10].
  combit CRM mobile Die Softwareschmiede combit bietet neben der klassischen Variante ihres CRM-Produkts auch eine für mobile Endgeräte zugeschnittene Variante. Mitarbeiter gelangen via Web-App auf eine speziell für Smartphones angepassten Oberfläche. Der Kunde kann die Anwendungsfläche außerdem unternehmensspezifisch verändert werden [11].
  Microsoft Dynamics CRM Im ersten Quartal 2013 gab Microsoft bekannt, dass Anwender sämtliche Funktionen der Software Microsoft Dynamics CRM auch auf verschiedenen mobilen Endgeräten nutzen können. Dazu nutzt Microsoft einen Cloudbasierten mobilen Service, der die Unterstützung für gängige Plattenformen wie Windows Phone 7, iPhone, iPad, Android und Blackberry garantiert. Managern, Vertriebs- und Service-Mitarbeiter sollen damit künftig jederzeit, überall und in Echtzeit auf Kundeninformationen zugreifen und diese wenn nötig aktualisieren können. Dank der flexiblen Entwicklungsumgebung von Microsoft Dynamics CRM ist es Unternehmen auch möglich, ihre eigenen CRM-Software-Addons für den mobilen Einsatz zu rüsten [21].

  Grafik 4 zeigt beispielhaft die App `CRM Sales` von SAP.
- **Ausgangslogistik**
  NOSTRA Logistics bietet mittels GPS-Tracking Echtzeit-Verfolgung von Kraftfahrzeugen. Es erlaubt außerdem die mobile Positionsbestimmung inklusivem Statusbericht über den aktuellen Zustand des Kraftfahrzeugs.

- **Marketing & Vertrieb**
  Oracle Mobile Sales Assistant Die Lösung „Oracle Mobile Sales Assistant“ ermöglicht den mobilen Zugriff auf Marketing- und Vertriebsdaten. Mitarbeiter der Verkaufsabteilung können somit mobil alle Kundendaten nutzen, bearbeiten und ad-hoc – beispielsweise nacheinem Telefonat mit dem Kunden Informationen hinterlegen.

Abbildung 4. Beispielansicht einer B2B-App (SAP CRM SAP) für das iPhone [2].

SAP Customer Briefing Für Vertriebsmitarbeiter bietet die App SAP Customer Briefing sowohl ortsunabhängigen Echtzeit-Zugriff auf Daten aus dem internen CRM/MRM System, als auch Zugriff auf externe Informationsquellen, wie Nachrichten oder Geschäftsberichte. Diese Informationen werden in einem leichtverständlichen Format präsentiert, so dass sich Mitarbeiter gezielt auf anstehende Kundenbesuche vorbereiten können. Grafik 5 zeigt die App in der iPad Version.

- **Kundenservice**
  Cubeware BI Neben dem Schwerpunkt CRM-Funktionen, gibt es bereits B2B Apps, die sich der Business Intelligence (BI) widmen. Die Firma Cubeware stellt mit Cubeware BI App eine mobile Lösung vor, um Managern den Zugang zu Reportings und Dashboards zu gewährleisten. Dashboards lassen sich dabei komplett oder in Einzelgrafiken herunterladen. Geschäftskennzahlen lassen sich mit Filter detailliert analysieren und weiterverarbeiten.

Im Folgenden werden Beispiel-Apps in den Kategorien der sekundären Aktivitäten nach Porter aufgezeigt:

- **Unternehmensinfrastruktur**
  Die Softwareschmiede forcont bietet die App *Business Lounge FX* an, mit der Mitarbeiter auch kollaborativ arbeiten können. Hauptaugenmerk ist das gemeinsame Treffen von Entscheidungen in Projekten. Dazu nutzt die App ein Umfrage-Tool, das die gemeinsame Abstimmung über Inhalte ermöglicht. Daneben verfügt die App über Dokumentenmanagementfunktionen [14].

  tablet2fax Auch dem Markt für Mobil-Security bzw. unternehmenskritische Mobil-Anwendungen widmen sich bereits einige Anbieter. Ferrari electronic,

Abbildung 5. Beispielansicht einer B2B-App (SAP Customer Briefing) für das iPad [2].

ein Spezialist für Unified-Communication, stellt eine App und dem begleitenden Service tablet2fax eine Möglichkeit bereit, rechtssichere Dokumentation von Schriftstücken und Fotos zu versenden. Dazu werden diese Schriftstücke mit einem mobilen Endgerät (Tablet oder Smartphone) abfotografiert. Anschließend wird das mobile Endgerät in ein Faxgerät gelegt. Der Bildschirm wird abgescannt und als Fax versendet [13].

SAP EHS Safety Issue Auch im Bereich Arbeitssicherheit bietet SAP eine App an. SAP EHS Safety Issue ist in der Lage, mögliche Schwachstellen zu dokumentieren. Dazu können Mitarbeiter eventuelle Sicherheitsrisiken in ihrem Arbeitsumfeld per Film, Foto oder Tonaufnahme festhalten und diese Informationen wenn nötig direkt an die Verantwortlichen des Facility-Managements übermitteln [24].

SAP Travel Expense Report Mit der App SAP Travel Expense Report können Mitarbeiter Spesenabrechnungen auf ihrem mobilen Endgerät erstellen. Dazu können Nutzer angefallene Kreditkartenbelege mit ihrem mobilen Endgerät abfotografieren und direkt auswerten lassen [24].

SAP Learning Assistant Die App SAP Learning Assistant ermöglicht einen einfachen Zugriff auf die SAP Plattform Learning Solutions. Damit können Mitarbeiter auch mobil direkt auf E-Learning-Inhalte zugreifen, Schulungen buchen oder sich persönliche Benachrichtigungen für auslaufende oder benötigte Qualifikationen anzeigen lassen [24].

IFS Notify Me Ähnlich wie SAP Travel Expense bietet der Softwarehersteller IFS mit IFS Notify Me, eine App zur vollständigen Automatisierung von Routineaufgaben/ Routineprozessen. Dazu zählen bspw. Genehmigungen von Bestellungen und Reisekosten. Mit der Hilfe der App sollen Manager

Zeit einsparen, um sich ihren Kernaufgaben zu widmen [18].

- **Personalwirtschaft**
  SAP Mobile Timesheet dient der mobilen Verwaltung Arbeitszeiten. Arbeitszeiten können von Mitarbeiter direkt eingegeben, überprüft und versendet werden [24].
  SAP HR Approvals ermöglicht die mobile von Bearbeitung von Anträgen. Sie ist an Manager des Human Ressource adressiert, so dass Genehmigungsprozesse schneller ablaufen können [24].

- **Technologieentwicklung**
  prinzipiell alle Apps Laut obiger Definition fallen nahzu alle B2B-Apps in diese Kategorie, da allein der Einsatz der mobilen Technologie unabhängig des Fachreichs zur Verbesserung von Arbeits- und Prozessabläufen führt. Die Definition dieser Kategorie ist folglich zu weitreichend, daher stellt Technologieentwicklung keine sinnvolle Kategorie und sollte deshalb ncht mitaufgeführt werden.

- **Beschaffung**
  SAP Mobile Procurement Die Lösung „SAP Mobile Procurement“ ermöglicht die mobile Beschaffung: Mitarbeiter können von unterwegs Waren und Dienstleistungen mit einem Handheld beschaffen. Hierbei stehen folgende Funktionen zur Verfügung: Einkaufswagen, Statusabfrage und Workflow-Inbox [24].

Die Kategorisierung von B2B-Apps nach Porters Wertschöpfungskette gestaltet sich zum Teil schwierig. Besonders die Zuweisung von multifunktionalen Apps in eine der wertschöpfenden Aktivitäten bedarf im Prinzip einer Priorisierung des beschriebenen Funktionsumfangs. Zudem erfolgt die Analyse des Funktionsumfangs manuell und ist somit für eine automatische Kategorisierung, in Anbetracht der stetig wachsenden Anzahl neuer B2B-App, eher untauglich.

4.4 Alternative Katalogisierungsansätze

Prinzipiell ist eine Katalogisierung nach jedem der oben genannten Merkmale möglich. Jede dieser Sichten kann zu einem sinnvollen Ergebnis aus Unternehmenssicht führen. Letztendlich entscheidet jedoch das Interesse und die Zielsetzung des Betrachters über die Sinnhaftigkeit der Katalogisierung. Da dies in der Literatur bislang weitestgehend unerforscht ist, soll hier eine Gegenüberstellung der Kategorisierungsalternativen in Kurzform vorgestellt werden.

Als Kriterien wurden zunächst vier Merkmale identifiziert, die intuitiv eine Eignung für eine Kategorisierung aufweisen:

1. *Automatisierbarkeit:* Hier ist die automatisierte Datensammlung und Aufbereitung gemeint. Beispielsweise ist eine Katalogisierung nach Sicherheitsaspekten oder Funktionsumfang nur manuell möglich, da die Hersteller der

Apps diese nur als Volltext in den App Stores bereitstellen.

2. *Eindeutigkeit:* Damit ist die eindeutige Zuordnung in eine Kategorie gemeint. Beispielsweise können Apps eindeutig nach Kundenbewertung (1-5 Sterne) katalogisiert werden.

3. *Zugänglichkeit:* Hier ist die öffentliche Zugänglichkeit der Information gemeint. Bei den Standardangaben wie Preis, Kundenbewertung, Betriebssystem usw. ist davon auszugehen, dass alle App Anbieter diese angeben. Bei sicherheitsrelevanten Aspekten unterscheidet sich die Vorgehensweise der App-Anbieter. Anders als Google Play Store stellt Apple App Store keine Informationen bzgl. Berechtigungen, über die eine App auf einem mobilen Endgerät verfügen darf, bereit.

4. *Relevanz für Unternehmen:* Damit soll zum Ausdruck gebracht werden, ob und wie eine Kategorisierung nach einem Kriterium, relevant für Unternehmen ist, die sich beispielsweise für den Einsatz von B2B-Apps interessieren.

Tabelle 4 und Tabelle 5 fasst die Ergebnisse der Katalogisierungsansätze zusammen.

Tabelle 4. Vergleich Katalogisierungskritierien Teil 1

	Funktionsumfang	**Preis**	**Kundenbewertung**	**Sicherheit**
Automatisierbarkeit	nein	ja	ja	nein
Eindeutigkeit	nein	ja	ja	nein
Zugänglichkeit	ja	ja	ja	zum Teil
Relevanz	hoch	mittel	mittel	hoch

Tabelle 5. Vergleich Katalogisierungskritieren Teil 2

	Tablet-Kompatibilität	**Branche**	**# Downloads**	**Kosten**
Automatisierbarkeit	ja	ja	ja	nein
Eindeutigkeit	ja	nein	ja	nein
Zugänglichkeit	ja	ja	nein	nein
Relevanz	mittel	hoch	niedrig	hoch

Die vorangegangene Auswertung soll lediglich als Hinweis dienen, nach dessen Ansatz eine alternative Kategorisierung von B2B-Apps vonstatten gehen könnte. Je nach Bedarf des Einsatzszenarios, empfiehlt es sich mehrere Kriterien heranzuziehen, um eine fundierte Entscheidung bzgl. der Eignung einer B2B-App für das Szenario zu ermitteln. Abschließend kann noch bemerkt werden, dass es im Bereich von B2B-Apps bisher keinen gemeinsamen Strukturen zu erkennen sind.

Handling und Design der Apps sind sehr inhomogen, so dass auch in der Hinsicht keine sinnvolle Kategorisierung möglich ist.

5 Empirische Untersuchung der Marktanteile

5.1 Auswertung der Marktanteile

Im folgenden Abschnitt soll anhand der im Kapitel 3 dargestellten Marktanalyse, eine grobe Analyse der Marktanteile hinsichtlich der identifizierten Kategorisierungsmerkmale durchgeführt werden. Die Hauptschwierigkeit bei der Untersuchung der Marktanteile einzelner App Kategorien bestand darin, scharfe Zahlen aus verlässlichen Quellen zu finden. Eine Kategorisierung von Apps in Anlehnung nach Porters Modell ist daher in der Regel nicht automatisierbar, deshalb wurde in der folgenden Auswertung 100 zufällig ausgewählte B2B-Apps aus dem Google Play Store verwendet. Um eine möglichst exakte Kategorisierung vornehmen zu können, musste der Funktionsumfang jeder App manuell erfasst und entsprechend analysiert werden. Falls eine App in mehrere Kategorien eingeordnet werden könnte, wurde sie entsprechend doppelt gezählt.

Tabelle 6 zeigt die Verteilung der ausgewählten B2B-Apps auf die verschiedenen Aktivitäten nach Porters Wertschöpfungskette

Tabelle 6. Emprische Auswertung der Marktanteile der App-Kategorien in absoluten Zahlen

Primäre Aktivitäten	**Anzahl Apps**
Eingangslogistik	18
Operation / Produktion	26
Ausgangslogistik	13
Marketing und Vertrieb	59
Kundendienst	33
Sekundäre Aktivitäten	**Anzahl Apps**
Unternehmensinfrastruktur	69
Personalwirtschaft	13
Technologieentwicklung	3
Beschaffung	33

Abbildung 6 zeigt, dass 40% der untersuchten Apps in die Kategorie „Marketing und Vertrieb“ eingeordnet werden können und damit die größte Kategorie im Bereich der primären Aktivitäten darstellt. Des Weiteren ist zu erkennen, dass 58% der untersuchten Apps in die Kategorie „Unternehmensinfrastrukur“ eingeordnet werden können und damit die größte Kategorie im Bereich der sekundären Aktivitäten darstellt. „Technologieentwicklung“ ist mit 3% mit großem Abstand die kleinste Kategorie.

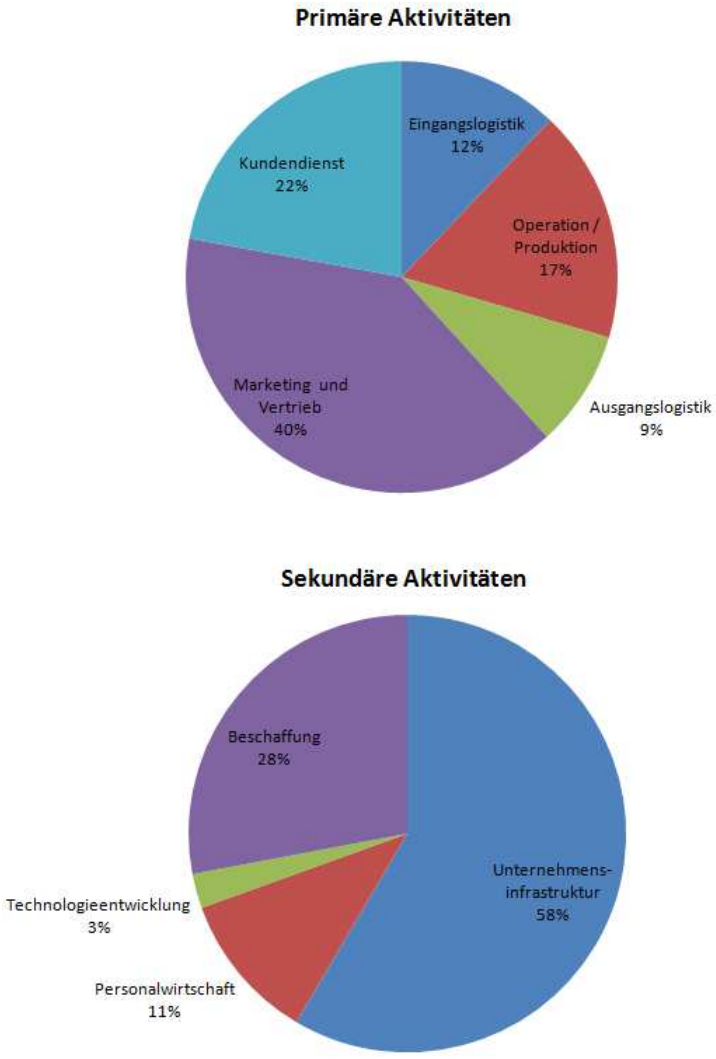


Abbildung 6. Prozentuale Verteilung von B2B-Apps auf die primären bzw. sekundären Aktivitäten nach Porters Wertschöpfungskette

6 Prognose und Ausblick

Eine vollständige Ausweitung der Kategorisierung auf den gesamten Bereich der mobilen B2B-Anwendung mit Einbeziehung diverser mobile Website-Diensten ist eher schwierig. Durch die hohe Anzahl von Merkmalen, die eine App auszeichnet ergeben sich sehr viele Möglichkeiten der Kategorisierung. Ein Unternehmen, das sich überlegt mobile B2B-Apps einzusetzen benötigt im Vorfeld sicherlich eine sorgfältige Sondierung des Angebots. Dies ist jedoch aktuelle nicht möglich, da es in allen App Stores an effektiven Filtermöglichkeiten mangelt. Dieser Mangel wird auch in Zukunft noch bestehen bleiben, da damit zu rechnen ist, dass es viele kleine bis mittlere Unternehmen geben wird, die hochspeziallisierte, stark kontextspezfische B2B-Apps anbieten. Nur damit können vollständig integrierte mobile Lösungen entstehen, die der Komplexität von B2B-Beziehungen gerecht werden. Damit ist jedoch die Schaffung eines einheitlichen Kategorisierungsstandards nur schwer durchzusetzen. Des Weiteren ist fraglich, trotz zahlreicher positiver Prognosen, inwiefern B2B-Apps den Anforderungen der Geschäftswelt gerecht werden können. Bisher gelingt es noch nicht, die komplizierte und zum Teil unübersichtliche Umstände des Geschäftsumfelds in Apps zu übertragen, da diese meist nur über einen limitierten Funktionsumfang verfügen. Darum könnte der Fokus von mobilen Anwendungen in Zukunft verstärkt auf mobile Websites (Web Apps) oder hybriden mobilen Anwendungen liegen, die in der Lage sind klassische Desktopaufgaben auch in die mobile Welt zu bringen.

7 Zusammenfassung

Unternehmen setzen vermehrt auf mobile Anwendungen im Geschäftsbereich. Insbesondere der Markt für mobile B2B-Apps wächst rasant. In den verschiedenen Fachabteilungen eines Unternehmens entstehen viele mögliche Einsatzszenarien, in denen Mitarbeiter von der Nutzung mobiler B2B-Apps profitieren können. Musterbeispiel sind Vertriebsmitarbeiter, die mobil per App auf zentral gelagerte Kundeninformationen zugreifen können, um sich optimal für den Einsatz beim Kunden zu präparieren. Auch im Arbeitsalltag bieten Apps Möglichkeiten Prozesse zu automatisieren und zu beschleunigen. Durch B2B-Apps können beispielsweise Genehmigungsprozesse und andere Routineaufgaben zeitsparend erledigt werden. Dabei ist die Vielfalt der Einsatzmöglichkeiten kaum noch zu überblicken. Für Unternehmen entsteht dadurch der Bedarf, geeignete Apps schnell und kostengünstig identifizieren zu können. Dafür ist es von Interesse verschiedene Kategorisiersmöglichkeiten aufzuzeigen, anhand dessen sich Unternehmen orientieren können. Auf dem Markt der B2C-Apps, existieren diese Möglichkeiten bereits. Marktplätze für B2B-Apps bieten diese Möglichkeit jedoch noch nicht. Es gibt zwar Ansätze verschiedener Anbieter, die den Unternehmen den Einsatz von B2B-Apps erleichtern, indem sie spezielle Purchasing-Programme bereitstellen. Diese helfen lediglich bei der Distribution von B2B-App jedoch nicht bei der Identifizierung geeigneter Apps.

Um eine geeignete Kategorisierung vornehmen zu können, mussten zunächst sämtliche Einzelmerkmale einer B2B-App identifizieren werden. Daraufhin erfolgt eine kritische Auseinandersetzung mit diesen Merkmalen, um die Tauglichkeit der jeweiligen Merkmale als Kategorisierungskritierium zu hinterfragen. Anschließend wurde anhand der Wertschöpfungskette nach Porter eine Möglichkeit aufgezeigt, wie B2B-Apps kategorisiert werden können, so dass eindeutig identifiziert werden kann, in welchem Bereich der Einsatz mobiler B2B-Apps zu einem Mehrwert für das Unternehmen führt. Im Anschluss darauf wurde versucht aktuelle B2B-Apps anhand des geschaffenen Schemas in Kategorien zu gruppieren. Des Weiteren wurde aufgezeigt, aus welchen Gründen andere alternative Ansätze der Kategorisierung in der Regel nicht geeignet sind. Zum Schluss wurde versucht, anhand öffentlich zugänglicher Informationen, eine erste, primitive Untersuchung der Marktanteile der verschiedenen Kategorien durchzuführen, um einen exemplarischen Blick die aktuelle Marktlage werfen.

Literatur

1. Apple: App store volume purchase program (zuletzt abgerufen am 15 Juni 2013), http://images.apple.com/de/business/docs/VPP_Business_Guide_DE.pdf
2. Apple: Apple app store (zuletzt abgerufen am 15 Juni 2013), https://itunes.apple.com/us/genre/ios-business/id6000?mt=8
3. Aventon: Aventon - logisticsone (zuletzt abgerufen am 15 Juni 2013), http://www.aventeon.com/index.php?id=44
4. Bassi, D.: Enterprise mobile apps (zuletzt abgerufen am 15 Juni 2013), http://www.research2guidance.com/enterprise-mobile-apps-200-000-in-stores-but-only-\-14-address-core-enterprise
5. Berger, S., Lehner, F.: Mobile b2b-anwendungen. Mobile and Collaborative Business pp. 85–94 (zuletzt abgerufen am 15 Juni 2013)
6. Bitkom: Apps and mobile services (zuletzt abgerufen am 11 Juni 2013), http://www.bitkom.org/files/documents/App_Studie_20110511_einzel.pdf
7. Bitkom: Das b2b-kundenverhalten im wandel - die herausforderungen im fokus (zuletzt abgerufen am 17 Juni 2013), http://www.bitkom.org/files/documents/App_Studie_20110511_einzel.pdf
8. Bitkom: Mobile anwendungen in der itk-branche (zuletzt abgerufen am 11 Juni 2013), http://www.bitkom.org/files/documents/App_Studie_20110511_einzel.pdf
9. Christmann, S., Hagenhoff, S., Schumann, M.: Mobiles internet im business-to-business-bereich-eine fallstudienuntersuchung. Arbeitsbericht des Instituts für Wirtschaftsinformatik, Universität Göttingen 1(4) (2009)
10. Cobra: Cobra - mobile crm (zuletzt abgerufen am 30 Juni 2013), http://www.cobra.de/?id=491
11. combit: combit crm mobile (zuletzt abgerufen am 30 Juni 2013), http://www.combit.net/crm-software/mobile-crm
12. Duffy, G., Dale, B.: E-commerce processes: a study of criticality. Industrial Management & Data Systems 102(8), 432–441 (2002)
13. ferrari electronic: tablet2fax service (zuletzt abgerufen am 29 Juni 2013), http://www.ferrari-electronic.de/

14. forcont: Business lounge fx (zuletzt abgerufen am 30 Juni 2013), http://www.forcont.de/produkte/business-lounge/erleben
15. Frauendorf, J., Kähm, E., Kleinaltenkamp, M.: Business-to-business markets–status quo and future trends. Journal of Business Market Management 1(1), 7–40 (2007)
16. Golem: App stores bringen umsatz von 25 milliarden us-dollar (zuletzt abgerufen am 15 Juni 2013), http://www.golem.de/news/apple_und_google_app_stores_bringen_umsatz_von_25_milliarden_us_dollar_1303_98221.html
17. Google: Google play private channel for google apps (zuletzt abgerufen am 19 Juni 2013), http://support.google.com/a/bin/answer.py?hl=en&answer=2494992
18. IFS-World: Ifs notify me (zuletzt abgerufen am 15 Juni 2013), https://cloud.ifsworld.com/Apps.aspx
19. IT-Business: Der richtige umgang mit mobile apps im b2b-umfeld (zuletzt abgerufen am 15 Juni 2013), http://www.it-business.de/mobility/apps/articles/394697/
20. Jutila, V., Kaukonen, H.P., Schmitgen, S.: Profi in wireless b2b. McKinsey Quarterly 1(1), 20–22 (2001)
21. Microsoft: Microsoft dynamics crm (zuletzt abgerufen am 30 Juni 2013), http://www.microsoft.com/de-de/dynamics/crm.aspx
22. One, D.: Data one mobile warehouse management (zuletzt abgerufen am 30 Juni 2013), http://www.dataone.de/DE/LOESUNGEN/DATAONEMOBILE/Seiten/default.aspx
23. Porter, M.E.: Competitive advantage: Creating and sustaining superior performance. SimonandSchuster. com (2008)
24. SAP: Sap store - mobile solution (zuletzt abgerufen am 15 Juni 2013), https://store.sap.com/sap/cpa/ui/resources/store/html/Solutions.html?pcntry=US&catID=MOB&sap-language=EN&_cp_id=id-1374143820899-0#All
25. Symposium/ITxpo, G.: Gartner identifies the top 10 strategic technology trends for 2013 (zuletzt abgerufen am 15 Juni 2013), http://www.gartner.com/newsroom/id/2209615
26. Verclas, S., Linnhoff-Popien, C.: Smart Mobile Apps. Springer DE (2011)
27. VisionMobile: Developer economics 2012 (zuletzt abgerufen am 15 Juni 2013), www.visionmobile.com/product/developer-economics-2012

Middleware für kontextsensitive Systeme

Ubiquitäre Systeme Seminar SS2013

Michael Jacoby[1] und Martin Alexander Neumann[2]

Karlsruhe Institute of Technology (KIT), Pervasive Computing Systems - TECO
[1]michael.jacoby@student.kit.edu
[2]mneumann@teco.edu

Zusammenfassung Aufgrund der steigenden Anzahl verfügbarer Sensoren und Mobilgeräte ist eine wachsende Nachfrage nach intelligenten, kontextsensitiven Anwendungen zu beobachten, die eine teilweise Automatisierung komplexer Aufgaben erlaubt. Ebenso ergeben sich durch die Verwendung von Kontext-Informationen innerhalb von Anwendungen auch völlig neue Anwendungsgebiete, die es zu erschließen gilt. Um die Entwicklung kontextsensitiver Anwendungen voranzutreiben ist es wichtig, die Entwicklung solcher Anwendungen möglichst einfach zu gestalten. Da kontextsensitive Anwendungen in der Regel in heterogenen und verteilten Umgebungen agieren, ist die Verwendung von middlewarebasierten Ansätzen besonders geeignet, denn diese konzentrieren sich auf die Unterstützung der Entwicklung von Anwendungen für ebensolche Umgebungen. Aus diesem Grund beschäftigt sich diese Arbeit mit den Anforderungen an kontextsensitiven Middlewaresystemen, die über die Anforderungen klassischer Middlewaresysteme hinaus gehen, sowie deren möglichen Ausprägungen. Es werden die Grundlagen von kontextsensitiven Systemen und Middleware erläutert und einige existierende Middlewaresysteme vorgestellt, die sehr unterschiedliche Ansätze verfolgen. Diese werden anhand der festgestellten Anforderungen verglichen und jeweils einem geeignetem Anwendungsgebiet zugeordnet.

1 Einleitung

Über die letzten Jahre hat die Verbreitung von mobilen Endgeräten stark zugenommen. Vor allem seit dem Aufkommen preisgünstiger und alltagstauglicher Smartphones findet man in nahezu jedem Haushalt ein solches Gerät. Der Trend bei der Entwicklung dieser Geräte folgt nicht nur dem, aus der Welt der Desktop-Computer und Laptops bekannten Paradigma, mehr Rechenleistung und Speicher bei weniger Energieverbrauch, sondern setzt auch verstärkt auf den Einsatz von unterschiedlichen Sensoren wie z.B. zur Messung von Beschleunigung, Helligkeit, Position, Magnetfeld und Temperatur. Parallel dazu steigt die Anzahl öffentlich und privat betriebener Sensoren zur Überwachung und Analyse der Umwelt, z.B. zur Vorhersage von Naturkatastrophen, Entdeckung von Waldbränden oder Steigerung des Ertrags in der Landwirtschaft. Diese steigende

Verfügbarkeit von Sensoren ermöglicht die Entwicklung neuartiger Software. So können Anwendungen entwickelt werden, die durch automatische Rekonfiguration basierend auf dem aktuellen Zustand der Umwelt einfacher zu bedienen sind und es dem Benutzer somit erlauben, sich stärker auf die zu erledigende Aufgabe zu konzentrieren, als sich mit der Konfiguration und den Besonderheiten der Anwendung zu beschäftigen. Ebenfalls ist es durch die Verfügbarkeit dieser Sensordaten möglich, neue, bisher nicht existierende Funktionen und Anwendungen zu entwickeln. Den für die Ausführung einer solchen Anwendung beobachteten Teil der Welt bezeichnet man als Kontext.

Die Verwendung von Middleware für kontextsensitive System ermöglicht die einheitliche Verwendung von heterogenen Endgeräten und Sensoren unter einem gemeinsamen Kontextmodell. Dies ist gerade für kontextsensitive Systeme wichtig, da diese in der Regel aus vielen heterogenen Komponenten bestehen. Daher ist das Ziel dieser Arbeit, verschiedene middlewarebasierte Ansätze für kontextsensitive Systeme hinsichtlich ihrer Architektur, Funktionsweise und möglichen Einsatzgebieten zu vergleichen.

Dazu folgt in diesem Abschnitt die Einführung und Definition der Begriffe Kontext, Kontextsensitivität und Middleware sowie eine Übersicht über die allgemeinen Anforderungen an Middlewaresysteme. Abschnitt 2 beschäftigt sich mit den speziellen Anforderungen an kontextsensitive Middlewaresysteme. In Abschnitt 3 werden vier existierende kontextsensitive Middlewaresysteme vorgestellt und ihre Stärken und Schwächen herausgearbeitet. Abschnitt 4 erläutert kurz die Bedeutung von kontextsensitiver Middleware für das Internet der Dinge. In Abschnitt 5 werden die vier vorgestellten Middlewaresysteme anhand ihrer Erfüllung der zuvor erarbeiteten Anforderungen verglichen, sowie jeweils mögliche Anwendungsgebiete aufgezeigt. Abschnitt 6 schließt diese Arbeit mit einer Zusammenfassung sowie einem Ausblick über die zukünftige Entwicklung und Bedeutung von kontextsensitiver Middleware.

1.1 Kontext

Zu dem Begriff Kontext finden sich viele Definitionen. Eine der allgemeinsten stellt die des Merriam-Webster's Collegiate Dictionary dar, die Kontext als „die in Zusammenhang stehenden Bedingungen unter denen etwas existiert oder auftritt“ (*„the interrelated conditions in which something exists or occurs“*) definiert. Diese Definition ist durch ihren hohen Grad an Abstraktion zwar sehr genau und allumfassend, dadurch hilft sie aber auch nicht das Konzept des Kontextes in der Informatik zu verstehen. Daher haben viele Forscher eigene Definitionen speziell für die Informatik formuliert. Dey und Abowd definieren Kontext wie folgt:

> *„Kontext ist jegliche Information, die verwendet werden kann, um den Zustand einer Entität zu beschreiben. Eine Entität ist dabei eine Person, ein Ort oder ein Objekt, das als relevant für die Interaktion zwischen Benutzer und Anwendung erachtet wird, Benutzer und Anwendung mit eingeschlossen.“* [8]

Diese Definition ist zwar immer noch recht allgemein gehalten, schränkt Kontext dabei aber schon explizit auf den Bereich der Informatik ein.

Oftmals wird versucht Kontext über Aufzählung und Klassifikation von Beispielen zu definieren. So nennt Schilit [23] viele Beispiele für Kontexte, wie z.B. Netzwerkverbindung, Kommunikationskosten und -bandbreite, in der Nähe befindliche Ressourcen wie Drucker oder Monitore, die Position des Benutzers, dessen gegenwärtige soziale Situation sowie Umgebungseigenschaften wie Helligkeit, Geräuschpegel und Temperatur. Gleichzeitig unterteilt er diese in die drei Klassen: informationstechnischer Kontext, Benutzerkontext und physischer Kontext. Chen et al. [5] schlagen eine Erweiterung um einen Zeitkontext sowie eine Kontexthistorie vor. Schmidt et al. [24] verwenden ebenfalls eine Kontexthistorie, gehen aber bei der Klassifikation andere Wege, indem sie Kontexte auf mehreren Abstraktionsebenen betrachten und somit eine Taxonomie zur Klassifikation von Kontexten erstellen. Diese ist in Abbildung 1 dargestellt.

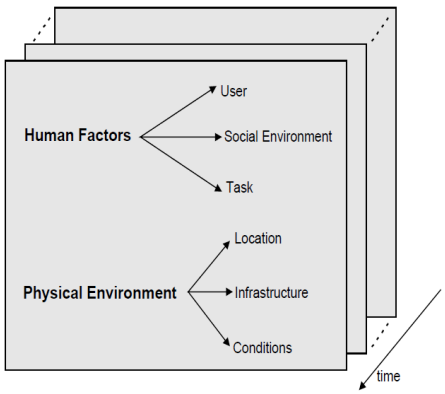


Abbildung 1. Kontext-Klassifikation nach [24].

1.2 Kontextsensitivität

Um kontextsensitive Anwendungen zu erstellen reicht eine Definition bzw. Klassifikation von Kontext nicht aus. Vielmehr benötigt man Konzepte um diesen effektiv in Anwendungen zu integrieren.

Schilit et al. [23] unterteilen Anwendungen anhand ihres Umgangs mit Kontextinformationen in folgende vier Klassen:

Proximate Selection
Hervorgehobene Darstellung von Objekten die gerade in der Nähe sind.

Automatic contextual Reconfiguration
Anpassung der Anwendung (z.B. durch Hinzufügen oder Entfernen von Komponenten) bei Kontextänderungen.

Contextual Information and Commands
Das Verhalten des gleichen Befehls wird durch den aktuellen Kontext bestimmt.

Context-triggered Actions
Einfache logische Anweisungen (wenn...dann...) welche die Reaktion der Anwendung auf Kontext festlegen.

Im Gegensatz zur Klassifikation von kontextsensitiven Anwendungen von Schilit et al. beschreibt Pascoe [21] eine Taxonomie von kontextsensitiven Funktionen. Trotz der unterschiedlichen Ansätze sind beide Klassifikationen nahezu deckungsgleich wie Dey und Abowd [8] zeigen. Außerdem präsentieren sie ihr eigenes Modell, das beide Ansätze kombiniert und Klassen von Funktionen umfasst, die kontextsensitive Anwendungen unterstützen können. Diese sind:

- Bereitstellen von Informationen und Diensten für einen Benutzer
- Automatisches Ausführen von Diensten
- Markieren von Daten mit Kontextinformationen

Chen et al. [5] unterteilen hingegen die Kontextklassen nochmals in aktive und passive Kontexte abhängig davon, wie diese in einer Anwendung verwendet werden. Ein Kontext bzw. eine Kontextklasse gilt als aktiv, sofern eine Anwendung automatisch auf die Änderung des Kontexts reagiert und ihr Verhalten anpasst. Als passiv gilt ein Kontext, wenn die Anwendung den geänderten Kontext lediglich anzeigt oder persistent speichert.

Die Vielzahl der verschiedenen Ansätze zeigt, wie viele neue Anwendungsmöglichkeiten sich durch kontextsensitive Systeme ergeben und wie unterschiedlich diese in Anforderungen und Funktion sein können.

1.3 Middleware

Middleware bezeichnet Softwaresysteme, die helfen sollen, mit der in verteilten Systemen inhärenten Komplexität und Heterogenität umzugehen und die Entwicklung verteilter Anwendungen einfacher zu gestalten. Angesiedelt zwischen Betriebssystem und Anwendungsschicht ist es ihre Aufgabe, verteilten Anwendungen Dienste zur Kommunikation und zum Datenaustausch bereit zu stellen und dabei von unterschiedlichen Betriebssystemen, der Netzwerkkommunikation, Datenformaten und vielen weiteren Aspekten zu abstrahieren und so eine schnellere und fehlerresistentere Entwicklung von verteilten Anwendungen zu gewährleisten.

1.3.1 Anforderungen Nach [11] [18] [10] bestehen folgende Anforderungen an eine Middleware:

Heterogenität
Abstraktion von verwendeter Hardware, Betriebssystem und Programmiersprache. Dies ermöglicht eine Kommunikation von Anwendungen zwischen

einer großen Anzahl von verschiedenen Geräten und ermöglicht die Verwendung von verschiedenen, dem Problem und der Zielplattform angepassten, Programmiersprachen.

Netzwerkkommunikation
Transparenz des verwendeten Netzwerkprotokolls und damit breite Unterstützung von gängigen Standards ermöglicht eine Kommunikation zwischen Geräten, die über unterschiedliche Verbindungstypen verbunden sind.

Skalierbarkeit
Skalierbarkeit bezeichnet die Fähigkeit, den Durchsatz bzw. die Leistungsfähigkeit eines Systems möglichst proportional zur Erhöhung der Rechenressourcen zu steigern. In einer zentralen Middlewarearchitektur ist diese durch die Leistung des Servers beschränkt. In einem verteilten System kann Skalierbarkeit erreicht werden, indem eine Anwendung auf mehrere Systeme verteilt wird. Damit dies ohne Änderungen an Architektur oder Code möglich ist müssen mehrere Aspekte des *ISO Open Distributed Processing (ODP)* Referenzmodells [1] beachtet werden. Um Skalierbarkeit zu erreichen muss eine Middleware Zugriffs-, Orts-, Migrations- und Replikationstransparenz unterstützen (Details siehe [10]).

Zuverlässigkeit
Unter Zuverlässigkeit versteht man die Zuverlässigkeit der Netzwerkkommunikation, da verschiedene Netzwerkprotokolle verschiedene Stufen von Zuverlässigkeit bieten. Daher muss die Middleware Fehlererkennung und -korrektur unterstützen. Des Weiteren kann Zuverlässigkeit auf Verbindungsebene durch die Umsetzung der ACID-Kriterien (*Atomicity, Consistency, Isolation, Durability*) um Zuverlässigkeit auf Transaktionsebene ergänzt werden. Außerdem umfasst Zuverlässigkeit auch die Erreichbarkeit des Systems, welche durch Replikation erhöht werden kann.

2 Kontextsensitive Middleware

Ziel einer kontextsensitiven Middleware ist es, ähnlich wie bei einer allgemeinen Middleware, das Erstellen von, in diesem Fall kontextsensitiven, Anwendungen zu erleichtern. Kontextsensitive Middleware unterscheidet sich von allgemeiner Middleware für verteilte Systeme in einigen Punkten. So steht bei kontextsensitiver Middleware das Sammeln, Aggregieren und Bereitstellen von Informationen im Vordergrund, während allgemeine Middleware den Fokus eher auf Kommunikation und verteiltem Rechnen setzt. Weitere spezielle Merkmale von kontextsensitiven verteilten Systemen sind die Art und Anzahl der verbundenen Systeme. Meist handelt es sich dabei um sehr viele, stark heterogene, oftmals kleine und mobile Systeme (z.B. Smartphones, Sensorknoten), die in der Regel über wenig Speicher-, Rechen- und Kommunikationsressourcen verfügen. Durch deren hohen Grad an Mobilität und der daraus resultierenden Unzuverlässigkeit der Kommunikation ergeben sich weitere Unterschiede. Weiterhin ist zu betrachten,

dass kontextbezogene Daten, wie z.B. Position, sehr sensible Daten darstellen können und ein höheres Schutzniveau fordern.

Die Frage, warum eine kontextsensitive Middleware einen Mehrwert darstellt und sich deren Entwicklung lohnt, beantwortet sich, indem man die alternativen Ansätze zur Entwicklung kontextsensitiver Systeme betrachtet. Der einfachste Weg ist, die Anbindung der Sensoren und das Kontextmodell direkt in der Anwendung zu implementieren. Dieser intuitive Ansatz birgt aber viele Nachteile in sich. So ist es beispielsweise nicht bzw. nur mit hohem Aufwand möglich die Sensorinformationen mit anderen Anwendungen auszutauschen. Zudem kann es zu Ressourcenkonflikten kommen, sobald mehrere Anwendungen gleichzeitig auf dieselben Sensoren zugreifen möchten. Außerdem ist der Entwicklungsaufwand für eine wartbare, umfangreichere, wiederverwendbare Anwendung relativ hoch, da der Sensorzugriff und die Verarbeitung der Sensordaten in jeder Anwendung neu programmiert werden müssen.

Um das Problem der Ressourcenkonflikte zu vermeiden, kann man das Betriebssystem um Funktionen zum Auslesen der Sensorwerte erweitern. Damit ist das Problem der redundanten Entwicklung jedoch nur teilweise gelöst, da die Verarbeitung der Sensordaten immer noch in jeder Anwendung neu implementiert werden muss. Ebenso entstehen dabei neue Risiken. Ein Fehler bei der Erweiterung des Betriebssystems kann zur Fehlfunktion dritter Programme oder gar zum Absturz des kompletten Systems führen. Außerdem wäre die Funktionalität dann nur auf diesem einen Betriebssystem verfügbar. Eine Abstraktionsstufe höher könnte man die Entwicklung einer API ansiedeln. Damit könnte man die Redundanz bei der Entwicklung minimieren, ist aber jedoch wieder an die Plattform bzw. Programmiersprache gebunden. Somit ist dieser Ansatz lediglich für Systeme geeignet, bei denen bekannt ist, dass nur genau eine Plattform verwendet wird. Dies ist jedoch bei größeren kontextsensitiven Systemen in der Regel nicht der Fall.

Aufgrund der aufgeführten Mängel der Alternativen stellt eine kontextsensitive Middleware die beste Möglichkeit dar, um die Entwicklung von kontextsensitiven Anwendungen zu vereinfachen.

2.1 Erweiterte Middlewareanforderungen

Nach [4] [22] ergeben sich folgende weitere Anforderungen an kontextsensitive Middleware:

Zuverlässigkeit & Mobilität
Aufgrund der hohen Mobilität der verwendeten Komponenten und der damit verbundenen Unzuverlässigkeit der Kommunikation müssen spezielle Netzwerk- und Routingprotokolle unterstützt werden, die unter diesen Bedingungen weiterhin eine den Anforderungen entsprechend zuverlässige Kommunikation gewährleisten.

Privatsphäre
Da Kontextinformationen oftmals sehr sensible Daten umfassen, muss dem Datenschutz der Benutzer Rechnung getragen werden.

Fehlertoleranz
Der Ausfall von Komponenten oder der Abbruch der Verbindung ist in diesem Umfeld nicht unwahrscheinlich und darf keine unerwarteten Folgen haben.

Einfache Installation und Konfiguration
Da die Middleware bzw. ein Teil davon ggf. auf Endgeräten installiert werden muss, ist eine einfache Installation und Konfiguration unerlässlich, um potentielle Nutzer nicht abzuschrecken.

Ressourcensparsamkeit & Heterogenität
Aufgrund der Vielzahl an verschiedenen mobilen und/oder mit Sensoren ausgestatteten Geräten, die oftmals über sehr geringe Rechen-, Speicher-, Energie- und Kommunikationsressourcen verfügen, treten diese beiden Anforderung im Vergleich zu einer normalen Middleware noch weiter in den Vordergrund.

Kontextmodell & einfach API
Damit Kontextinformation in heterogenen Systemen verarbeitet werden können, müssen diese ein gemeinsames Verständnis haben, was Kontext ist und wie dieser darzustellen ist. Darum ist die Definition eines gemeinsam benutzten Kontextmodells notwendig. Um eine einfache Entwicklung neuer kontextsensitiver Anwendungen zu gewährleisten, muss das System eine einfach zu verstehende API anbieten. Genauere Anforderungen an diese beiden Punkte werden im folgenden Abschnitt erläutert.

2.2 Kontextmodell & API

Damit der Austausch von Kontextinformationen zwischen allen Geräten möglich ist, ist eine einheitliche Beschreibung von Syntax und Semantik der auszutauschenden Daten nötig. Eine solche Beschreibung bezeichnet man als Kontextmodell. Neben dem Datenaustausch zwischen Systemen ermöglicht es auch das Reasoning, also das maschinelle Ziehen von Schlüssen auf Basis bekanntem Wissens, auf den gemeinsam gesammelten Kontextinformationen durch die Middleware. Daher kann man auch sagen, eine Middleware kapselt ihr Kontextmodell und beschreibt damit das Datenformat, das für die Kommunikation mit ihr zu verwenden ist. Ein solches Kontextmodell sollte nach [15] und [25] folgende Anforderungen erfüllen:

Einheitlichkeit
Alle verschiedenen Kontexte sollten nach einem einheitlichen Schema modelliert werden. Dadurch wird eine einheitliche Verarbeitung der Kontextdaten sowie Reasoning ermöglicht.

Maschinenlesbarkeit
Um eine automatische Verarbeitung sowie Reasoning zu ermöglichen ist ein maschinenlesbares Kontextmodell unerlässlich.

Erweiterbarkeit
Da nie alle möglichen Eigenschaften von Kontexten antizipiert werden können, ist eine Erweiterbarkeit notwendig, um die zukünftige Bedürfnisse befriedigen zu können.

Standardisierte Repräsentation
Aufgrund der starken Heterogenität in Hard- wie auch in Software ist eine kanonische Repräsentation unabdingbar.

De-/Komponierbarkeit
Ein System sollte in die Lage versetzt werden, nur den Teil des Kontextmodells kennen zu müssen, der für es interessant ist. Ferner sollte das Nachladen (lokal sowie von anderen Systemen) von Modellteilen möglich sein.

Austauschbarkeit
Um De-/Komponierbarkeit zu gewährleisten, ist die Austauschbarkeit der Daten über Systemgrenzen hinweg notwendig. Dies fordert eine Serialisierbarkeit des Modells bzw. von Teilmodellen.

Metadaten-Modell
Ein Metadaten-Modell ist bei der Verarbeitung von Kontextinformationen von großem Nutzen, da es Informationen über die Informationen bereitstellt. Diese Informationen können Auskunft darüber geben, woher die Kontextinformationen kommen oder wie zuverlässig sie sind. Gerade letzteres ist von großem Interesse und Nutzen auf dem Gebiet der Kontextsensitivität, da hierbei meist Sensordaten verarbeitet werden, die inhärent unsicher und verrauscht sind.

Anforderungen an eine Schnittstelle einer kontextsensitiven Middleware werden indirekt in [22] dargestellt und lassen sich wie folgt beschreiben:

Unterstützung von CRUD-Operationen auf Sensordaten
CRUD steht für **C**reate, **R**ead, **U**pdate, **D**elete und bezeichnet die vier Grundoperationen für persistenten Speicher. Diese Funktionen sind notwendig, um einerseits Sensordaten mit Hilfe der Middleware zu speichern, andererseits damit Anwendungen auf diese gespeicherten Informationen zugreifen können.

Benachrichtigung von Systemen beim Eintreten von Ereignissen von Interesse
Da viele mit der Middleware verbundene Systeme beschränkte Hardware-, Kommunikations- und Energieressourcen besitzen ist es notwendig, eine Möglichkeit zu schaffen, diese Geräte von rechen- und kommunikationsintensiven Aufgaben zu befreien und diese in die Middleware zu verlagern. Dazu ist ein Benachrichtigungssystem nötig, welches die verbundenen Systeme bei dem Auftreten eines entsprechenden Ereignisses informiert.

Abfrage von Metadaten
Um bei einem solch verteiltem System Zugriff auf alle verfügbaren Kontextinformationen zu gewährleisten ist es notwendig, eine Möglichkeit zum Auffinden von Informationen ohne Kenntnis über deren Existenz zu schaffen. Dies lässt sich über eine Abfragemöglichkeit von Metadaten realisieren.

3 Vergleich existierender Middlewaresysteme

In diesem Abschnitt werden vier existierende Middlewaresysteme vorgestellt und ihre Funktionsweise näher beschrieben. Zu jedem System werden die Vor- und Nachteile genannt und erklärt. In Kapitel 5 werden diese in Bezug auf einige ausgewählte Anforderungen aus Abschnitt 1.3.1 sowie Abschnitt 2 gegenübergestellt und bewertet.

3.1 LIME

LIME (Linda in a Mobile Environment) [20] ist eine Java-basierte Middleware deren Ziel es ist, durch minimale Abstraktion schnelle und zuverlässige Entwicklung von mobilen Anwendungen zu gewährleisten und Entwickler von der Komplexität der Kommunikation zwischen Host und Agenten zu befreien. Dabei stellt ein Host einen mobilen Container für Agenten dar, der eine Netzwerkverbindung sowie eine Laufzeitumgebung für Agenten bereitstellt. Die Basis von LIME bildet, wie der Name schon sagt, Linda [13], eine Programmiersprache für verteiltes Rechnen.

Das Konzept von Linda ist, die Interprozesskommunikation über einen gemeinsam genutzten, persistenten Speicher, TupleSpace genannt, zu realisieren. Er erlaubt den gleichzeitigen Zugriff von mehreren Prozessen und bietet folgende Operationen an:

out(t)
Fügt das Tupel t dem TupleSpace hinzu.

in(p)
Entfernt ein Tupel, das dem Pattern p entspricht, aus dem TupleSpace und wird blockierend ausgeführt. **inp(p)** wird asynchron ausgeführt und liefert ε falls kein passendes Tupel zum Zeitpunkt der Abfrage existiert.

rd(p)
Gibt ein Tupel, das dem Pattern p entspricht, aus dem TupleSpace zurück ohne es zu löschen (blockierend). Die asynchrone Version heißt **rdp(p)** und liefert ebenfalls ε falls kein solches Tupel existiert.

LIME stellt eine Koordinationsschicht für logisch und physisch verteilte Systeme bereit, indem es das Kommunikationsmodell von Linda übernimmt und erweitert. Anstelle des gemeinsam genutzten, persistenten Linda-TupleSpace bietet

LIME einen globalen, virtuellen TupleSpace, der durch den regelbasierten spontanen Zusammenschluss aller zu einem Zeitpunkt verbundenen Agenten entsteht (siehe Abbildung 2). Die auf den Agenten existierenden Teile des globalen virtuellen TupleSpace werden *interface tuple space* (ITS) genannt. Ein ITS hat einen Namen (eindeutig pro Agent), gehört ausschließlich zu einem Agenten und beinhaltet alle Tuple die der Agent anderen zugänglich machen möchte, sowie alle Tuple der gleichnamigen ITS aller Agenten auf demselben Host. Agenten sind verbunden, solange sie sich auf demselben Host oder zwei verbundenen Hosts befinden. Hosts sind verbunden, solange sie sich nahe genug beieinander befinden, dass eine Kommunikation möglich ist. Bauen zwei bisher nicht verbundene Hosts eine Verbindung auf, so werden die ITS der Agenten des einen Host mit denen der Agenten des anderen Host in einer atomaren Aktion, Kopplung (*engagement*) genannt, vereinigt. Der Verbindungsabbau läuft analog und wird Abkopplung (*disengagement*) genannt. Den Zusammenschluss der ITS aller Agenten eines Hosts bezeichnet man als *host-level* TupleSpace. Den Zusammenschluss der host-level TupleSpaces aller verbundenen Host wiederum als *federated* TupleSpace.

Eine Anfrage mit **in(p)** oder **rd(p)** auf einen host-level oder federated TupleSpace liefert nicht-deterministisch ein beliebiges Tupel aus einem beliebigen ITS zurück. Da dies nicht immer gewünscht ist und manche Anwendungen genauer über die Herkunft oder den Verbleib ihrer Daten bestimmen möchten, hat LIME die Linda-Schnittstelle um Funktionen mit den Tuple-Speicherort-Parametern λ bzw. ω erweitert, wobei ω den aktuellen Speicherort des Tupels und λ dessen Ziel-Speicherort beschreibt.

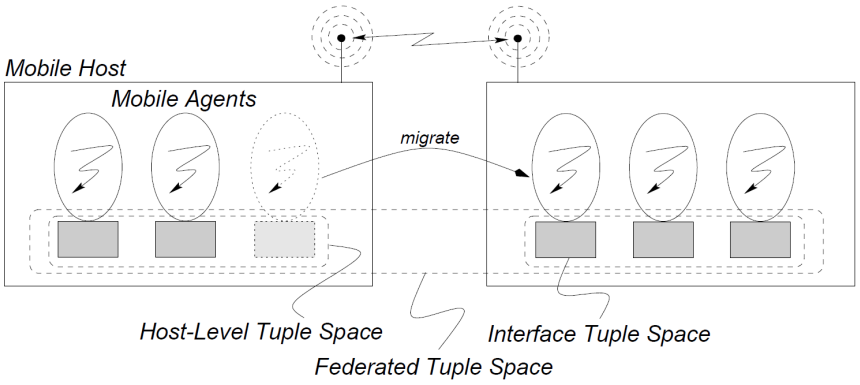


Abbildung 2. Lime TupleSpaces [20].

Die Operation **out[λ](t)** fügt nun das Tupel t zuerst in den lokalen ITS ein (mit der Information, dass es an der Stelle λ gespeichert werden soll) und überträgt es an es λ, sobald eine Verbindung besteht. Wurde es erfolgreich übertragen,

wird es aus dem ITS des Aufrufers entfernt. Dabei ist λ die ID eines Agenten oder Hosts. Des Weiteren führt LIME auch die Befehle **in[**ω,λ**](p)** bzw. **inp[**ω,λ**](p)** und **rd[**ω,λ**](p)** bzw. **rdp[**ω,λ**](p)** ein, die es erlauben, die Operation **in(p)** bzw. **rd(p)** auf einem durch die beiden Parameter ω und λ eingeschränkten Bereich des federated TupleSpace auszuführen (Details siehe [20]). Um der hohen Dynamik eines mobilen ad-hoc Netzwerkes Rechnung zu tragen, führt LIME das Konzept der Reaktion $\mathcal{R}$(s,p) ein, wobei s ein Codefragment ist, das die Aktion beschreibt, die ausgeführt werden soll, sobald ein Tupel das das Pattern p erfüllt, gefunden wird. Die Semantik ist dabei angelehnt an Mobile UNITY [19]. Nach jeder Operation auf dem TupleSpace wird solange zufällig jeweils eine Reaktion aus der Menge der registrierten Reaktionen gezogen und ausgewertet bis nur noch nicht erfüllte Reaktionen verbleiben. Die syntaktische Definition einer Reaktion in LIME lautet **$\mathcal{R}$[**ω, λ**](s,p)** wobei die Parameter ω und λ analog zu **in[**ω,λ**](p)** bzw. **inp[**ω,λ**](p)** zu verstehen sind.

Um die IDs von Agenten und Hosts zu ermitteln und um auf Verbindungsaufbau sowie -trennung reagieren zu können, stellt LIME einen systemverwalteten, schreibgeschützten TupleSpace, genannt *LimeSystem*, zur Verfügung. Dieser beinhaltet Informationen zu den verbundenen Agenten und Hosts, welche TupleSpaces sie jeweils öffentlich anbieten und welcher Agent zu welchem Host gehört.

Auf Ebene der Softwarearchitektur bietet LIME mit LighTS eine Schnittstelle zur Abstraktion der verwendeten TupleSpace Implementierung. Neben der in LighTS integrierten Implementierung gibt es auch einen Adapter für TSpaces von IBM [17].

Neben der hier beschriebenen LIME-Version gibt es noch zwei weitere, die speziell für den Einsatz in Sensornetzen angepasst wurden: TinyLIME [7] sowie TeenyLIME [6]. Da die Kommunikation bei LIME direkt auf Socket-Ebene geschieht und somit kein RMI notwendig ist, ist die Verwendung auf verschiedenen Plattformen recht einfach. Bisher getestet wurden Windows, Linux, MacOSX sowie PDAs mit PersonalJava. Als Netzwerkprotokolle wurden Ethernet sowie IEEE 802.11 verwendet.

Stärken und Schwächen

Die Stärken von LIME sind vor allem sein sehr simples Interface, dass die Benutzung für unerfahrene Entwickler einfacher macht sowie seine Unterstützung von vielen Plattformen aufgrund der sehr geringen Dateigröße als auch den spezialisierten Versionen (TinyLIME, TeenyLIME) für Sensorknoten. Durch die in LighTS enthaltene Schnittstelle, die eine völlige Abstraktion des verwendeten TupleSpace ermöglicht, wird eine einfache Erweiterung sichergestellt. Die Kommunikation über ad-hoc-basierte Netzwerke erlaubt eine komplett dezentrale Struktur, wodurch die Installation und Inbetriebnahme des Systems aufgrund des fehlenden Servers vereinfacht wird. Die Verwendung von ad-hoc Kommunikation bringt allerdings auch Probleme mit sich. So ist der Zugriff auf Daten anderer Hosts nur möglich, wenn diese gerade in Reichweite sind. Die Modellierung von Kontexten als Tuple in LIME kann man als Schlüssel-Wert basiertes Kontextmodell verstehen, bei dem der Schlüssel der Position des Wertes im Tu-

pel entspricht. Dieses Kontextmodell ist sehr primitiv und hat den Nachteil, dass alle Hosts, die Kontextinformationen austauschen wollen, diese in der gleichen Art und Weise in Tupeln ausdrücken müssen. Ferner unterstützt LIME mit LighTS auch nur ein extrem simples pattern-matching. DieseM Problemen haben sich Balzarotti et al. angenommen und beschreiben in [3] wie man LighTS für die Entwicklung von kontextsensitiven Systemen erweitern kann, z.B. durch die Einführung von Fuzzylogik. Fuzzylogik dient dabei meist der Modellierung von Unsicherheiten und erlaubt eine Unterteilung eines Intervalls in mehrere, sich teilweise überschneidende Klassen, auch Fuzzy-Mengen oder Fuzzy-Sets genannt. Durch das primitive Kontextmodell werden Entwickler leider dazu gezwungen, Teile des Kontextmodells in ihrer Anwendung zu realisieren, was einer sauberen Trennung von Anwendungslogik und Kontextverarbeitung im Wege steht. Tabelle 1 zeigt die Stärken und Schwächen von LIME in einer Übersicht.

Stärken	Schwächen
Sehr einfaches Interface	Zugriff auf Daten anderer Agenten nur bei Konnektivität
Unterstützung vieler Plattformen	Sehr primitives Kontextmodell
Leichtgewichtig	Standardmäßig nur Unterstützung von einfachen pattern-matching Algorithmen
Erweiterbar	Keine saubere Trennung zwischen Anwendungslogik und Kontextverarbeitung
Dezentral	
Einfache Installation	

Tabelle 1. Stärken und Schwächen von LIME.

3.2 CASS

CASS (Context-Awareness Sub-Structure) [12] ist eine serverbasierte Middleware. Ihr Ziel ist die Unterstützung der Entwickler von kontextsensitiven Anwendungen auf kleinen und mobilen Geräten. Um dieses Ziel zu erreichen bietet sie eine klare Trennung zwischen Anwendungslogik und Kontextverarbeitung. Dies ermöglicht eine Änderung der Kontextinterpretation während der Laufzeit. Dadurch ist es möglich den Benutzer selbst zur Laufzeit entscheiden zu lassen, nach welchen Regeln die Anwendung mit Kontext interagiert. Weitere Merkmale von CASS sind die Erweiterbarkeit bezüglich neuer Kontext-Klassen sowie die Unterteilung der Kontext-Zustandsräume. Außerdem bietet CASS Unterstützung von vielen low-level Sensoren sowie anderen Kontext-Datenquellen. Da für die meisten Anwendungen die high-level Kontexte am wichtigsten und interessantesten sind [9], unterstützt CASS kontextbasierte Inferenz mit Hilfe des Forward-Chaining-Algorithmus.

Abbildung 3 zeigt die serverbasierte Architektur der CASS-Middleware. Unter Sensorknoten werden dabei alle Geräte subsumiert, die lokal mit einem oder

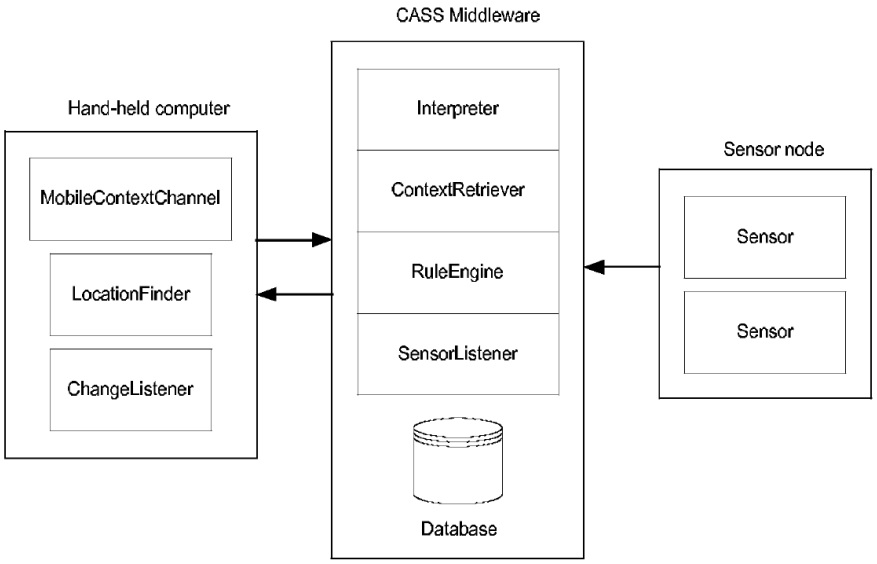


Abbildung 3. CASS Architektur [2].

mehreren Sensoren und per Netzwerk mit dem Server verbunden sind. Sie können mobil sowie stationär sein und senden ihre gemessenen Sensordaten an den Server.

Um Schwankungen in der Verbindungsqualität zu begegnen verwenden sie einen lokalen Cache-Speicher, der die Daten für den Fall von Verbindungsproblemen zwischenspeichert. *Hand-held computer* stellen die Plattform dar, auf der die kontextsensitive Anwendung läuft. Sie kommunizieren über WLAN mit dem Server und können mit Hilfe der Klasse *ChangeListener* auf Kontext-Änderungen reagieren. Der Server, in der Abbildung als *CASS Middleware* bezeichnet, besteht aus einer relationalen Datenbank, der Klasse *SensorListener*, die auf Aktualisierungen der Sensoren reagiert, sowie der Klasse *ContextRetriever*, deren Aufgabe darin besteht, den Zugriff auf gespeicherte Kontextinformationen zu kapseln.

In der Datenbank werden neben Kontext-, Anwendungs- und Benutzerdaten auch noch Domänendaten in Form von (Inferenz-)Regeln gespeichert. Diese Speicherung der Regeln, zusammen mit einer Schnittstelle zur Manipulation dieser, erlaubt die Rekonfiguration der Inferenz zur Laufzeit durch die Anwendung oder den Benutzer. Weiterhin umfasst der Server noch die beiden Klassen *RuleEngine* und *Interpreter* welche den Inferenz-Mechanismus implementieren.

Stärken und Schwächen

CASS zeigt seine Stärken bei der sauberen Trennung von Anwendungslogik und Kontextverarbeitung sowie bei der Inferenz von höherwertigen Kontexten. Die

Regeln für die Inferenz durch Vorwärtsverkettung (forward-chaining) werden in der Datenbank des Server gespeichert und können von der Anwendung oder dem Benutzer zur Laufzeit durch eine einfache Schnittstelle verändert werden, wodurch eine dynamische Adaption des Verhaltens möglich ist. Durch die Verwendung eines zentralen Servers ergeben sich allerdings auch einige Nachteile. So wird dadurch zum einen die Skalierfähigkeit stark eingeschränkt und zum anderen ist eine Netzwerkverbindung für jegliche Operation notwendig. Bemerkbar macht sich das vor allem in einem Szenario, in dem ein mobiles Gerät gleichzeitig als Sensor als auch als Konsument von Kontextinformationen auftritt. Hier ist ohne Netzwerkverbindung selbst die Verarbeitung von lokalen Sensorinformationen nicht möglich. Des Weiteren resultiert durch die ausschließlich serverseitige Verarbeitung von Sensorwerten ein sehr hohes Kommunikationsvolumen zwischen Sensoren und Server, was bei einer Anbindung über mobiles Internet (UMTS, GPRS, o.Ä.) zu hohen Kosten und Performanceverlust führen kann sowie den Energieverbrauch des Sensors bzw. Sensorknotens erhöht. Tabelle 2 zeigt die Stärken und Schwächen von CASS in einer Übersicht.

Stärken	**Schwächen**
Unterstützung von Kontext-Inferenz	Schlechte Skalierbarkeit
Dynamische Anpassung der Inferenzregeln zur Laufzeit möglich	Hohes Kommunikationsvolumen zwischen Sensoren und Server
Saubere Trennung zwischen Anwendungslogik und Kontextverarbeitung	Netzwerkverbindung immer notwendig

Tabelle 2. Stärken und Schwächen von CASS.

3.3 Hydrogen Context-Framework

Das Hydrogen Context-Framework [16] (im Folgenden als Hydrogen bezeichnet) beschreibt eine 3-schichtige Architektur. Im Unterschied zu den anderen hier vorgestellten Systemen laufen bei dieser Middleware alle drei Schichten auf demselben Gerät, dem mobilen Endgerät. Abbildung 4 gibt einen recht detaillierten Überblick über die Architektur und Funktionsweise von Hydrogen. Die unterste Schicht, der *Adaptor Layer*, bildet eine Abstraktion zu den verwendeten Sensoren. Dies ist notwendig um Ressourcenkonflikte bei zeitgleichem Zugriff mehrerer Anwendungen auf denselben Sensor zu vermeiden.
Die mittlere Schicht, der *Management Layer*, besteht aus dem *ContextServer*. Dessen Aufgabe ist es, die Kontextdaten zu speichern und eine Schnittstelle zur Abfrage dieser Daten anzubieten. Unterstützt werden neben Abfragen auch Benachrichtigungen, die über das Publish/Subscribe-Muster realisiert sind. Ferner besteht die Möglichkeit durch Anpassung des *ContextServer* den Austausch zwischen verschiedenen mobilen Geräten, die miteinander verbunden sind, zu realisieren (z.B. über Peer-to-Peer). Dies ist aber nach Hofer et al. [16] nicht

explizit Bestandteil von Hydrogen, sondern vielmehr eine bisher noch nicht realisierte Erweiterung, deren Umsetzung neue Fragestellungen nach Sicherheit und Zuverlässigkeit der Daten aufwirft.

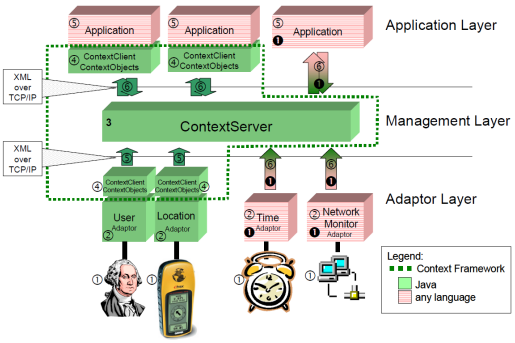


Abbildung 4. Hydrogen Architektur [16].

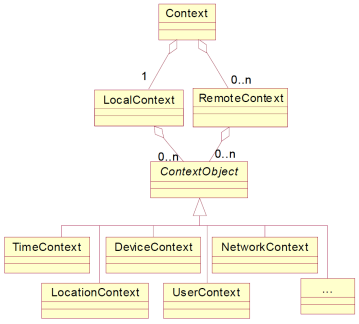


Abbildung 5. Hydrogen Kontextmodell [16].

Die oberste Schicht ist die Anwendungsschicht (*Application Layer*) und besteht lediglich aus Hilfsklassen zum Zugriff auf den *Management Layer* sowie Anwendungen (die aber nicht zur Middleware gehören). Die Kommunikation zwischen den Schichten ist durch XML über TCP/IP sowie alternativ über serialisierte Java-Objekte realisiert. Das erlaubt die Implementierung von Anwendungen und Adaptoren in einer beliebigen Programmiersprache.

Hydrogen verwendet ein objektorientiertes Kontextmodell. Abbildung 5 zeigt die dazugehörige Klassenhierarchie in UML-Notation. Das Kontextmodell ist bereits vorausschauend geplant worden und umfasst neben einem lokalen Kontext auch eine Menge von Remote-Kontexten. Dabei wird jeder Kontexttyp als Unterklasse von *ContextObject* modelliert.

Stärken und Schwächen

Durch die Entkopplung der einzelnen Schichten mit Hilfe der Kommunikation über XML via TCP/IP erreicht Hydrogen fast eine vollständige Unabhängigkeit von Plattform und Programmiersprache. Bis auf den *ContextServer* können alle Komponenten in einer beliebigen Sprache und Plattform implementiert werden, sofern diese XML und TCP/IP unterstützen. Außerdem erlaubt diese Entkopplung das Betreiben der einzelnen Schichten auf unterschiedlichen Geräten. Eine weitere Stärke von Hydrogen stellt die bereits teilweise angelegte Schnittstelle für eine P2P-Kommunikation mit anderen Geräten dar. Dass diese Funktionalität jedoch nicht enthalten ist stellt eine Schwäche des Systems dar, genauso wie das Fehlen einer Kontexthistorie. Eine weitere Schwäche ist das sehr minimale Kontextmodell. Dieses kann zwar erweitert werden, erfordert aber die Anpassung aller Schichten sowie deren Neukompilierung. Eine Möglichkeit zur

dynamischen Erweiterung des Kontextmodells wäre hier vorteilhafter. Tabelle 3 zeigt die Stärken und Schwächen von Hydrogen in einer Übersicht.

Stärken	Schwächen
Dezentral bzw. lokal	Kein fertiges Konzept für Kommunikation zwischen Geräten
(fast) vollständige Plattform- und Programmiersprachenunabhängigkeit	Keine Kontexthistorie
Erweiterungsmöglichkeit um P2P-Kommunikation.	Sehr minimales Kontextmodell
	Keine dynamische Erweiterung des Kontextmodells möglich

Tabelle 3. Stärken und Schwächen von Hydrogen.

3.4 Service-Oriented Context-Aware Middleware (SOCAM)

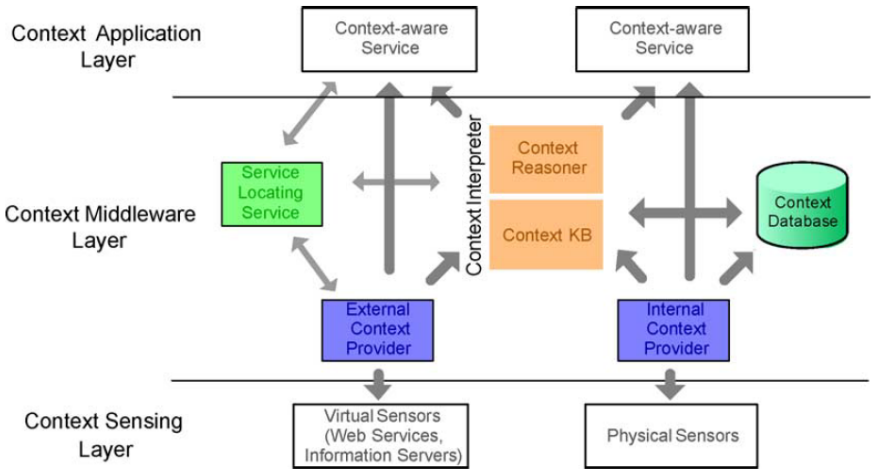


Abbildung 6. SOCAM Architektur [14].

SOCAM beschreibt ebenfalls eine 3-schichtige Architektur wie in Abbildung 6 zu sehen. Im Gegensatz zu Hydrogen umfasst die SOCAM Middleware keine Aufgaben der oberen und unteren Schicht und implementiert somit lediglich die mittlere Schicht, den *Context Middleware Layer*. Ein weiterer Unterschied zu allen bisher angeführten Middlewaresystemen besteht im Kontextmodell. SOCAM verwendet hierbei das Konzept der Ontologie, auf das später genauer eingegangen wird.

Auf unterster Ebene der Middleware stehen die *Context Provider*. Sie bilden die Schnittstelle zu verschiedenen Sensoren und wandeln deren Daten in ein Ontologie-konformes Datenformat um. Außerdem registrieren sie sich beim *Service Locating Service*, der eine Art Verzeichnis für *Context Provider* darstellt. Die durch die *Context Provider* gewonnen Daten werden durch viele Komponenten verwendet. Zum einen werden sie in einer Datenbank (*Context Database*) gespeichert. Zum anderen sind sie durch ihre Registrierung beim *Service Locating Service* für alle anderen Komponenten über die dort hinterlegten Metainformationen auffindbar. Somit stellt der *Service Locating Service* den zentralen Ansprechpunkt für alle Komponenten dar. Er liefert zu einem Informationsbedürfnis eine Referenz auf den entsprechenden *Context Provider* oder *Context Interpreter* zurück, der wiederum die nötigen Informationen bereitstellt. Die Kommunikation mit ihm erfolgt über TCP/IP und Java RMI.Zu den Aufgaben eines *Context Interpreters* zählt das Inferieren von high-level Kontexten sowie die Wahrung der Konsistenz der Kontextdaten und die Auflösung von Kontext-Konflikten. Er besteht aus einer Wissensdatenbank (*Context KB*) und einem *Context Reasoner*. Die Wissensdatenbank bietet eine Schnittstelle für die gängigen CRUD-Operationen für Kontextdaten und enthält eine Kontext-Ontologie sowie die dazugehörenden Instanzen. Der *Context Reasoner* unterstützt momentan zwei Arten des Reasonings: Ontologie-Reasoning und regelbasiertes Reasoning. Generell unterstützt er die Verwendung verschiedener Reasoning-Arten sowie deren Konfiguration zur Laufzeit durch den Benutzer und Entwickler.

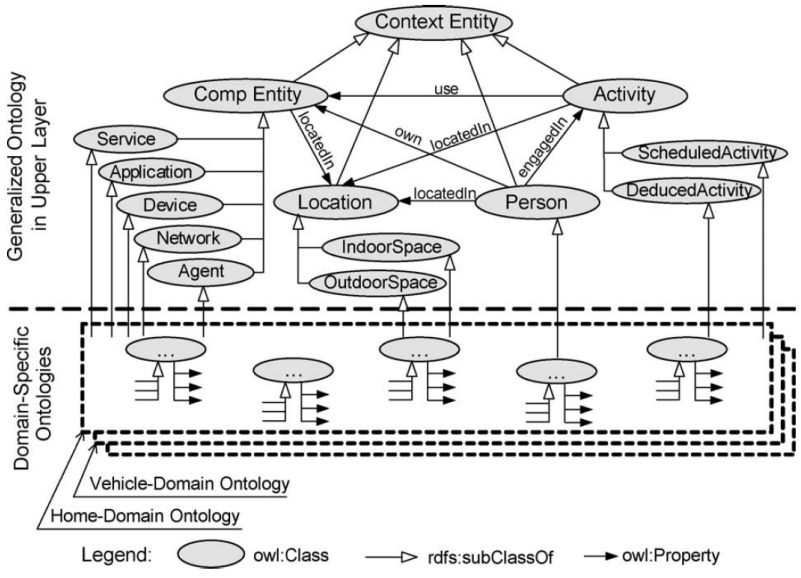


Abbildung 7. In SOCAM verwendete Ontologien und ihre Hierarchie [14].

Wie in Abbildung 7 zu sehen, arbeitet SOCAM mit einer zweistufigen Ontologie-Hierarchie. Auf der oberen Schicht befindet sich eine generalisierte Ontologie, die die allgemeinen Konzepte von kontextsensitiven Systemen und Kontext beschreibt. Auf der unteren Ebene befinden sich domänenspezifische Ontologien die beschreiben, wie genau die allgemeinen Konzepte in der jeweiligen Domäne zu verwenden sind und welche Eigenschaften sie haben. Dies ermöglicht die dynamische Einbindung und Auskopplung der zu jedem Zeitpunkt interessanten domänenspezifischen Ontologien. Dies führt neben verringertem Rechen- und Speicheraufwand auch zu deutlicher höherer Flexibilität.

Stärken und Schwächen
Eine der größten Stärken von SOCAM ist sein ontologiebasiertes Kontextmodell. Durch die Verwendung von Ontologien ist es extrem mächtig und auch problemlos dynamisch anpassbar. Die Verwendung von Ontologien liefert quasi umsonst Unterstützung zur Abfrage von Metadaten zum Auffinden von unbekannten Sensoren bzw. Kontexten. Außerdem bieten Ontologien ein für Reasoning sehr geeignetes Datenformat. SOCAM bietet, wie bereits erwähnt, neben dem ontologiebasierten Reasoning auch ein regelbasiertes Reasoning an, welches durch den Benutzer zur Laufzeit konfiguriert werden kann. Eine weitere Stärke von SOCAM ist die gute Skalierbarkeit, da von allen Komponenten außer dem *Service Locating Service* und der Datenbank mehrere Instanzen existieren können.

Tabelle 4 zeigt die Stärken und Schwächen von SOCAM in einer Übersicht.

Stärken	Schwächen
Sehr mächtiges Kontextmodell	Komplexes Kontextmodell
Dynamische Erweiterung des Kontextmodells möglich	Höhere Anforderungen an Endgeräte
Unterstützt zwei Arten von Reasoning	
Kontexthistorie	
Gute Skalierbarkeit	
Unterstützt Abfrage von Metadaten	

Tabelle 4. Stärken und Schwächen von SOCAM.

4 Internet der Dinge und kontextsensitive Middleware

Das Internet der Dinge, auch mit IoT (Internet of Things) abgekürzt, ist keine Technologie oder Anwendung sondern vielmehr ein Paradigma, dass viele verschiedene Technologien, z.B. aus den Gebieten Kommunikation, Cloud, Sensorhardware/-software und Semantik, umfasst und vereinigt. Oftmals wird es auch als der nächste Schritt in der Evolution des Internets gesehen wie in

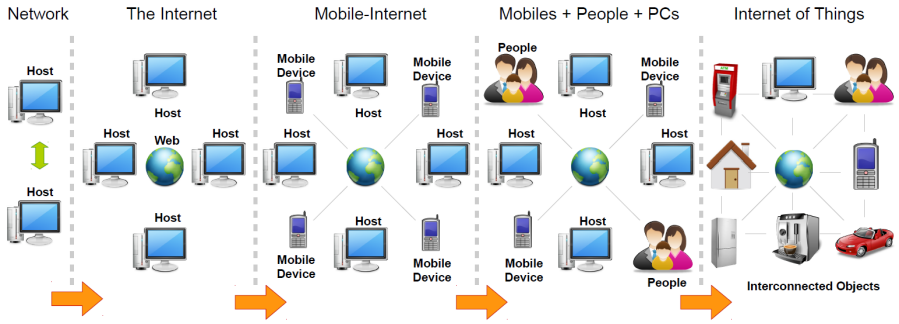


Abbildung 8. Die Evolution des Internets in fünf Stufen nach [22].

Abbildung 8 dargestellt. Ziel des IoT ist es, eine „bessere Welt“ für die Menschen zu schaffen, indem die Dinge um uns herum „intelligent“ gemacht werden und uns beim alltäglichen Leben von sich aus unterstützen. Dies kann ebenfalls durch direkte Kommunikation und Interaktion zwischen diesen Dingen, auch Smart Objects oder Interconnected Objects (ICOs) genannt, passieren. Dabei spielen Kontextinformationen eine wesentliche Rolle, denn sie stellen die einzige Möglichkeit für ICOs dar, um Informationen über den Zustand der Welt autonom zu sammeln um angemessen drauf zu reagieren. Somit wird deutlich, dass Kontextsensitivität eine wesentliche Rolle im IoT spielt. Aufgrund der immensen Anzahl von heterogenen Sensoren und Akteuren/ICOs im IoT ist eine sehr gut skalierbare Middleware notwendig. Abbildung 9 zeigt die wichtige Rolle von kontextsensitiver Middleware im IoT.

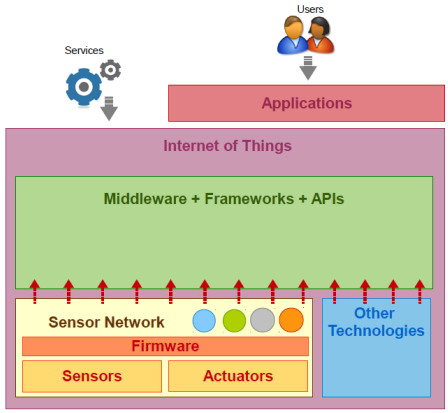


Abbildung 9. Zusammenhang zwischen IoT und Middleware [22].

5 Diskussion

Die vier vorgestellten Middlewaresysteme unterscheiden sich in ihrer Architektur und dem daraus resultierenden Anwendungsgebiet deutlich. In diesem Abschnitt soll deshalb zunächst ein Vergleich anhand der zuvor festgestellten Anforderungen erfolgen. Anschließend werden anhand der Stärken und Schwächen der Systeme mögliche Anwendungsgebiete aufgezeigt.

5.1 Vergleich

Tabelle 5 vergleicht die vier vorgestellten Middlewaresysteme anhand ausgewählter Anforderungen aus Abschnitt 1.3.1 sowie Abschnitt 2. Dabei entspricht + bzw. ++ einer guten bzw. sehr guten, o einer neutralen sowie - bzw. -- einer schlechten bzw. sehr schlechten Erfüllung der Anforderung.

	LIME	CASS	Hydrogen	SOCAM
Fehlertoleranz	++	--	o	+
Einfachheit	++	+	+	-
Heterogenität	++	o	+	+
Zuverlässigkeit & Mobilität	++	-	(+)	+
Kontextmodell	Tupel	Objekte	Objekte	Ontologie
Standardisierung	--	o	+	++
Erweiterbarkeit	+	o	o	++
Metadaten	o	--	--	++
Abfrage von Metadaten	o	--	--	++
Skalierbarkeit				
Anzahl Sensoren	o	-	-	++
Anzahl Nutzer	o	-	-	++
Kontexthistorie	o	++	--	++
Reasoning	--	+	--	++
Architektur	verteilt, ad-hoc	zentralisiert, Server	verteilt, P2P	verteilt, mit 2 zentralen Komponenten

Tabelle 5. Vergleich der Middlewaresysteme.(++/+ entspricht (sehr) gut, o neutral, --/- (sehr) schlecht)

Fehlertoleranz
LIME ist aufgrund des komplett dezentralen Aufbaus und der ad-hoc Kommunikation kaum fehleranfällig, SOCAM steht dem jedoch kaum etwas nach da nur zwei Komponenten zentral sind. Bei Hydrogen hängt die Fehlertoleranz stark von der physischen Verteilung der einzelnen Schichten ab - sind diese alle auf dem selben Gerät, so ist die Fehlertoleranz recht hoch. Sind

sie jedoch auf verschiedenen Geräten liegt das gleich Problem wie bei CASS vor, und zwar, dass durchgängig eine Netzwerkverbindung bestehen muss.

Einfachheit
LIME kann vor allem durch seine sehr minimalistische und einfache API punkten, CASS und Hydrogen durch ihre intuitiven objektorientierten Kontextmodelle. SOCAM hingegen ist durch sein ontologiebasiertes Kontextmodell recht kompliziert.

Heterogenität
Aufgrund der verschiedenen existierenden Versionen von LIME für unterschiedliche Plattformen wird es hier sehr gut bewertet. Hydrogen und SOCAM setzen bei der Kommunikation auf offene, plattformunabhängige Datenformate und Protokolle und sind daher prinzipiell offen für die Anbindung anderer Plattformen. Bei CASS sind hierzu keine Informationen zu finden. Da die Architektur jedoch bei geeigneter Wahl von Datenformaten und Protokollen Heterogenität unterstützt, wird dies hier neutral bewertet.

Zuverlässigkeit & Mobilität
Hier schneidet CASS am schlechtesten ab, da es eine durchgehende Netzwerkverbindung zwischen Sensor und Middleware erfordert. Hydrogen hingegen besitzt von Haus aus keine Möglichkeit zur Kommunikation zwischen zwei Instanzen der Middleware und bietet lediglich eine Schnittstelle zur eigenen Implementierung. SOCAM unterstützt Mobilität durch das dynamische Auffinden von Komponenten und Kontexten durch den *Service Locating Service*. LIME setzt durch die dezentrale ad-hoc basierte Kommunikation die Anforderung am besten und umfassendsten um.

Kontextmodell
LIME bietet mit seinem Tupel-basierten Kontextmodell ein sehr einfaches und erweiterbares Kontextmodell. Aufgrund der Einfachheit gibt es dabei keine Standardisierung. Eine Speicherung von Metadaten ist prinzipiell möglich, jedoch kann dies nicht explizit durch das Modell dargestellt werden.
CASS und Hydrogen verwenden beide ein objektorientiertes Kontextmodell. Dieses umfasst ebenfalls keine Metadaten, ist jedoch erweiterbar, allerdings nicht ohne Neukompilierung des Systems. Im Punkt Standardisierung unterschieden sie sich. Hydrogen verwendet in XML serialisierte Objekte über HTTP, bei CASS gibt es dazu keine Angaben.
SOCAM erfüllt durch die Verwendung eines ontologiebasierten Kontextmodells alle hier aufgeführten Anforderungen sehr gut.

Abfrage von Metadaten
Die Abfrage von Metadaten bei CASS und Hydrogen ist nicht möglich, da diese keine Unterstützung von Metadaten liefern. Die LIME ist diese Abfrage möglich, jedoch sehr umständlich, da aufgrund des simplen Kontextmodells keine komplexen Abfragen möglich sind. SOCAM wird hier wiederum mit sehr gut bewertet, da es die Abfrage von Metadaten über SPARQL, SPARQL

Protocol and RDF Query Language - eine SQL-ähnliche Abfragesprache für RDF-Daten, unterstützt.

Skalierbarkeit
LIME hat durch sein Kommunikationsmodell über ad-hoc Netzwerke Probleme bei der Skalierung in der Anzahl der Nutzer, da bei vielen Benutzern die Wahrscheinlichkeit steigt, dass diese sich bewegen und somit alte Verbindungen abbrechen und neue entstehen. Dabei muss jeweils der Datenbestand aller Teilnehmer angeglichen werden, was einen großen Aufwand darstellen kann. In der Anzahl der Sensoren ist die Skalierfähigkeit ähnlich, da mit der Anzahl der Sensoren auch die Anzahl der Daten steigen, und somit mehr Daten zu synchronisieren sind. Bei CASS ist der zentrale Server der Flaschenhals, den jeder Sensor und Nutzer passieren muss. Daher wurde die Skalierbarkeit mit schlecht bewertet. Das Gleiche gilt auch für Hydrogen. SOCAM erreicht durch die größtenteils dezentrale Organisation und der Möglichkeit, mehrere Instanzen von nahezu allen Komponenten gleichzeitig zu betreiben eine sehr gute Skalierung in Bezug auf Sensoren sowie Nutzer.

Kontexthistorie
Bei SOCAM und CASS ist eine Kontexthistorie explizit vorhanden. Bei LIME kann diese theoretisch durch die Verwendung der Operation *rd(p)* anstelle von *in(p)* realisiert werden. Bei Hydrogen ist hingegen keine Kontexthistorie vorgesehen und implementiert. Eine manuelle Erweiterung ist denkbar, allerdings nicht ohne Neukompilierung des Systems.

Reasoning
LIME und Hydrogen unterstützten kein Reasoning. CASS unterstützt Reasoning, allerdings ausschließlich regelbasiertes Reasoning. Da sich die (Inferenz-) Regeln zur Laufzeit dynamisch ändern lassen wird CASS hier mit gut bewertet. SOCAM erfüllt die Anforderung sehr gut, da neben dem regelbasierten Reasoning noch Ontologie-Reasoning unterstützt wird. Zudem können weitere Reasoning-Arten sehr einfach implementiert werden.

5.2 Anwendungsgebiete

LIME
Aufgrund seiner ad-hoc basierten Kommunikation ist bei LIME die Position der Nutzer immer implizit Kontext, da die Kommunikation nur zwischen Geräten funktioniert, die sich in Reichweite befinden. Daher ist das Anwendungsgebiet von LIME auch auf Anwendungen beschränkt, die mit diesem Umstand umgehen können. Es bieten sich hier besonders Anwendungen an, bei denen soziale und oder räumliche Interaktion im Vordergrund stehen, wie z.B. Augmented-Reality-Spiele.

CASS
Aufgrund des Bedarfs einer dauerhaften Verbindung und dem hohen Datentransfers ist CASS eher für den Einsatz von Sensoren mit einer kabelgebundenen und/oder lokalen Netzwerkverbindung geeignet. Durch die Unterstützung

von regelbasiertem Reasoning ist CASS beispielsweise sehr geeignet für Smart Homes.

Hydrogen
Hydrogen ist durch seine mangelnde Kommunikationsfähigkeit lediglich dafür geeignet, als zusätzliche Abstraktionseben auf einem Mobilgerät zwischen Sensoren und Anwendungen zu dienen, ähnlich den bereits für verschiedene Smartphone-Betriebssysteme existierenden Sensor- oder Kontext-APIs wie z.B. Google Location API.

SOCAM
SOCAM ist aufgrund seiner redundanten und dezentralen Architektur und seines komplexen Kontextmodells sehr vielseitig einsetzbar. So wurde es beispielsweise bereits im Smart Home- sowie im Fahrzeugbereich eingesetzt. Außerdem könnte diese Architektur mit einigen kleineren Änderungen auch Teil des IoT werden. Grundsätzlich sollte es dort eingesetzt werden, wo die anderen Systeme an ihre Grenzen in Bezug auf Skalierfähigkeit und/oder Kontextmodellierung geraten. Für Szenarien mit einer kleinen Nutzergruppe oder lediglich mobilen Endgeräten ist SOCAM aufgrund seiner Komplexität nicht zu empfehlen.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Definition von Kontext- und Kontextsensitivität gegeben. Es wurde der Begriff Middleware erklärt, sowie Anforderungen an eine Middleware dargestellt. In Abschnitt 2 wurden die erweiterten Anforderungen an kontextsensitive Middleware aufgezählt und erläutert. Danach wurden vier existierende Middlewaresysteme detailliert vorgestellt und ihre Stärken und Schwächen aufgezeigt. In Abschnitt 4 wurde eine Brücke zum IoT geschlagen und der Zusammenhang zwischen kontextsensitiver Middleware und dem IoT dargestellt. Im letzten Abschnitt wurden die vier vorgestellten Middlewaresysteme anhand der zuvor erarbeiteten Anforderungen verglichen und die jeweiligen Anwendungsgebiete vorgestellt.

Dabei hat sich gezeigt, dass jede Architektur ihre Stärken und Einsatzgebiete, und somit auch ihre Daseinsberechtigung hat. In zukünftigen IoT-Szenarien werden sich wohl aufgrund der Heterogenität und Diversität der Anwendungen mehrere dieser Architekturen wiederfinden. Außerdem ist zu erwarten, dass durch die Vielzahl der aktuellen Forschungsprojekte vom Thema IoT neue, leistungsstärkere Architekturen entstehen werden.

Literatur

1. Open distributed processing - reference model, 1998.
2. Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

3. Davide Balzarotti, Paolo Costa, and Gian Pietro Picco. The lights tuple space framework and its customization for context-aware applications. *Web Intelligence and Agent Systems*, 5(2):215–231, 2007.
4. Marco Bessi and Leonardo Bruni. A survey about context-aware middleware.
5. Guanling Chen, David Kotz, et al. A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.
6. Paolo Costa, Luca Mottola, Amy L Murphy, and Gian Pietro Picco. Teenylime: transiently shared tuple space middleware for wireless sensor networks. In *Proceedings of the international workshop on Middleware for sensor networks*, pages 43–48. ACM, 2006.
7. Carlo Curino, Matteo Giani, Marco Giorgetta, Alessandro Giusti, Amy L Murphy, and Gian Pietro Picco. Tinylime: Bridging mobile and sensor networks through middleware. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 61–72. IEEE, 2005.
8. Anind K Dey and Gregory D Abowd. Towards a better understanding of context and context-awareness.
9. Anind K Dey, Gregory D Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2):97–166, 2001.
10. Wolfgang Emmerich. *Engineering distributed objects*. John Wiley & Sons, 2000.
11. Wolfgang Emmerich. Software engineering and middleware: a roadmap. In *Proceedings of the Conference on The future of Software engineering*, pages 117–129. ACM, 2000.
12. Patrick Fahy and Siobhan Clarke. Cass–a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*. Citeseer, 2004.
13. David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.
14. Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, 28(1):1–18, 2005.
15. Albert Held, Sven Buchholz, and Alexander Schill. Modeling of context information for pervasive computing applications. In *Procceding of the World Multiconference on Systemics, Cybernetics and Informatics*, 2002.
16. Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices-the hydrogen approach. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 10–pp. IEEE, 2003.
17. Tobin J Lehman, Stephen W McLaughry, and Peter Wyckoff. T spaces: The next wave. In *System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on*, pages 9–pp. IEEE, 1999.
18. Anna Liu. Gathering middleware requirements. In *Information Networking, 2001. Proceedings. 15th International Conference on*, pages 81–86. IEEE, 2001.
19. Peter J McCann and G-C Roman. Compositional programming abstractions for mobile computing. *Software Engineering, IEEE Transactions on*, 24(2):97–110, 1998.
20. Amy L Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(3):279–328, 2006.

21. Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 92. IEEE Computer Society, 1998.
22. Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. 2013.
23. Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 1994.
24. Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.
25. Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004.

Modeling Dynamics of Ubiquitous Systems

Ubiquitäre Systeme Seminar SS2013

Yan Bi[2] and Martin Alexander Neumann[1]

[1] Karlsruhe Institute of Technology (KIT), Pervasive Computing Systems - TECO
mneumann@teco.edu
[2] biyan727@gmail.com

Abstract. This paper will focus on Modeling Dynamics of Ubiquitous Computing Systems. While many innovations during the last decade made progress in human computer interaction technology, the next critical step is to make computing everywhere and invisible to people. Therefore, Ubiquitous Computing plays an increasingly important role and modeling, as one of the most useful system development methods, will be discussed in the paper. To satisfy the dynamic requirements of Ubiquitous Computing, the characteristics of complex distributed and context-dependent systems will be reviewed. Many modeling approaches are applied to model systems, such as Simulation Models, Model-Checking, and Execution-Based Model-Checking. After the introduction, a discussion about their differences and benefits is provided according to the typical development tools. The development and future of Ubiquitous Computing are promising.

1 Introduction

Nowadays humanity's daily lives are increasingly dependent on computer systems. In the early years, only specialists used computers to calculate the problems for their research and it was unavailable to the general public. Time flies by, and computers are becoming everywhere available.
It took still a long time to learn about computers in spite of its popularization and computers caught too much attention of users. Mark Weiser noticed the point and thought it was the development direction of modern science and technology. Computers should be more easily accessible and usable. Computing systems seem to disappear by becoming ubiquitous. He called this future world "Ubiquitous Computing" (short form:"Ubicomp")[1].

1.1 Motivation

A lot of researchers have been involved in the development of Ubicomp. They tried many different ways to make computing system smaller so that people can bring it everywhere, and made excellent progress in enabling people to enjoy the convenient without noticing its existence. Model Driven Architecture (MDA), as an innovative software development methodology, is one of the important approaches, which can simplify the whole process of developing ubiquitous systems. Numerous new modeling methods and tools appear continually over the last few years. Each one has its own advantages and disadvantages just like a coin with two sides.
The thesis selects three most common methods and makes a comparison among them. The main contribution of this paper is that the summary of various modeling approaches may simplify the way of finding a suitable method in a given use case.

1.2 Outline

This chapter as an introduction gives the motivation and overview of this paper.
Chapter 2 is basic knowledge. It starts with the rapid development of Ubiquitous Computing, and then its dynamics and characteristics, including conceptions of complex distribution and context dependence being briefly introduced.
In chapter 3, Model Driven Design, including its definition, structure and development processes, is reviewed and then three modeling methods of Ubiquitous Computing, i.e. simulator, Model-Checker and Execution-based model-checker, are explained and some tools are elaborated through a series of examples and compared with each other.
Brief demonstrations of a same case using different tools are given in chapter 4. After that some conclusions are drawn from the paper in the last but not least chapter.

2 Basic Knowledge

Developing ubiquitous systems are complicated and includes many processes. Many researchers apply Software Engineering Methodology to structure, plan, and control these processes and adapt many models to the changed conditions, each characterizing a system of methods and principles for developing Ubiquitous Computing Systems.
In this chapter the history of Ubicomp will be given and the requirements of

developing ubiquitous systems will be discussed. The diverse classifications of context will be listed.

2.1 Ubiquitous Computing Systems

Mark Weiser, who mentioned the concept–"Ubicomp" in his paper "the Computer for the 21st Century" in 1991, is considered to be the father of ubiquitous computing. Actually, the concept was coined in 1988 during his work at Xerox PARC in the United States.[2]
The main purpose of Ubiquitous Computing, which is also described as pervasive computing, is to intensify computer use by enabling the communication among the computers throughout the physical environment, but ensuring the invisibility to the user.[2]

Phase I	The Mainframe Era:	Many people use a common computer.
Phase II	The PC Era	Everyone has his own computer.
Transition	The Internet and Distributed Computing	The internet brings together elements of the mainframe era and the PC era.
Phase III	The UC Era	Lots of computers can be found everywhere and share each of us.

Table 1. four phase of computing trends

Just like the automatic revolving doors. Years ago you should push the door, if you want to get in. Now there are sensors! The door can receive the instructions from computing centre, which is processing information from embedded sensors to determine whether that is passer-by or visitor. The most advanced automated doors can even detect the visitor is permitted to enter or not. In this way, people can step through the "intelligent door" without paying any attention to its existence.
The most profound technologies in Dr. Weiser's eye were those that disappear. He thought that they weaved themselves into the fabric of everyday life until they were indistinguishable from it[1]. To describe the history and future of computing system, he defined two historical phases, and a transition, and also predicted a new era.[2] They are given in Table 1.

In the middle of the last century, since the advent of computers, or rather calculation instruments, many specialists shared one computer, because it was expensive, rare and hard to use. With the development of science and technology, many people owned a PC, before the 21th century was coming. After transition, the third wave of computing–Ubicomp has appeared as predicted by Dr. Weiser, which has begun in 2005 and would last for 15 years. [2]. With the emergence and development of cloud computing, networking, social network concepts and other modern technologies, the future of Ubicomp is promising and the ubiquity will be the future of science and technology.
First of all, we need to study its characteristics, in order to develop and improve ubicomp systems.

2.2 Dynamics of Ubiquitous Systems

If the current output, or rather the current state, of the system is related to the past history as well as the present values of the input variables, the system will be called dynamic system[3]. Ubicomp system should react to the present status and can't ignore the historical record. Ubicomp system is briefly definitely dynamic. Ubicomp technologies aim at collecting, providing and analysing dynamic information from the environment and provide related people a suitable assistance with as little noise as possible.
The requirements of ubiquitous systems are very different from conventional software systems. Usually a world is represented by three main constituents (see Figure 1):[4]

1. Physical world: Everything can be described by physical theories, such as temperature, humidity, weight, etc. The ubiquitous system is related to the physical world, so physical properties must be involved, whose values represent states of an object or a system. Suitable measurable physical properties will be selected depending on system requirements and observed to be part of the dynamic information of a ubicomp system.

2. Digital world: Software components and applications, which provide information resources for ubiquitous computing, can be found everywhere. And every computing component are connected to other components of the same system with local area network(LAN) or wide area network(WAN). The small mobile phone in your pocket can nowadays even manage the office system as well as monitor the air conditioner at home over the Internet.

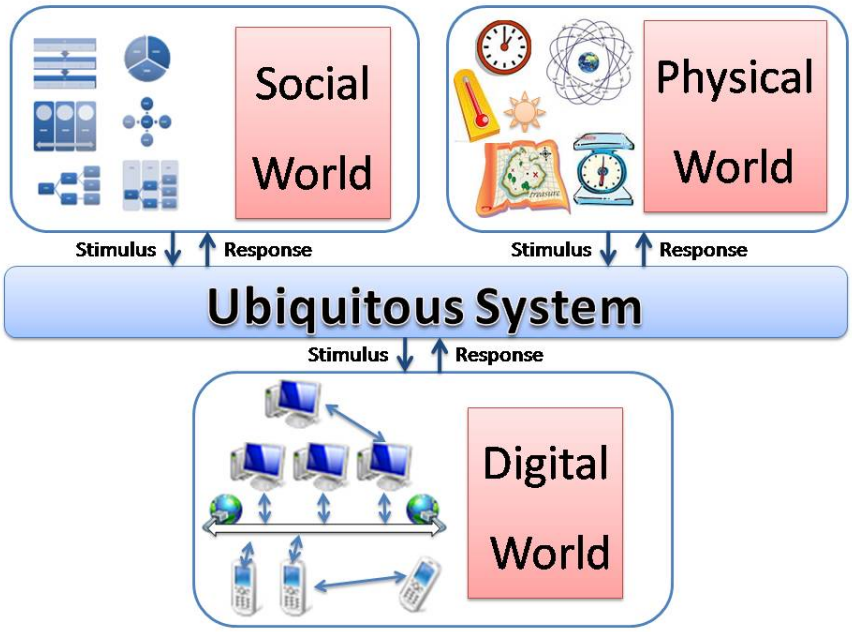


Fig. 1. Ubiquitous System and its environment (based on [4])

3. Social world: The core of computing systems is to help users, so the social environment around users is more and more crucial. Business Process Management (BPM) is one of the components of the social world, which defines the business processes and promotes business effectiveness and efficiency. In addition, the mental state of people and the assigned tasks may also have an effect on the computing system.

The Figure 1 is a description of a ubiquitous system and its environment. The goal of ubiquitous systems is to gather the information from the above mentioned three main constituents, viz. to observe its environment and, more importantly, to make decisions based on these observations. According to this idea, the entire world around us is the data base resource of an information system and the ubicomp system is a network of intelligent computing devices, which constantly monitor its state. In order to design such a ubicomp system, the whole environment should be defined as a composition of suitable models.
A ubicomp system interacts with its environment (Physical, Digital and Social) depending on stimulus-response behaviour. The distribution of the Digital World (Figure 1)turn ubicomp systems into being complex distributed. And

specific to ubicomp systems is that the response depends on stimuli and on the state of the system context because of its dynamical feature. These properties have to be adequately addressed during the entire design process of ubicomp systems.

2.3 Complex Distributed

Because a ubicomp system is ubiquitous, it must consist of several distributed computing devices (see Digital World of Figure 1) which are located in different places and can communicate through Wireless Sensor Network (WSN). These computing devices interact with each other in order to achieve a common goal. Ubicomp system is complex distributed, because it needs to exchange information among its devices and share resource with other systems. But characteristics of ubicomp systems differ significantly from traditional distributed systems, which is marked by authorized networks, static network topology and absolute position (fixed), the features of Ubiquitous Computing can be listed as follows: ad-hoc networks, highly dynamic topology, relative position (dynamic, physical proximity).
Coordination requires communication. The efficiency of collaboration is one of the main examination indexes.

2.4 Context Dependent

Context-aware technology is a hot issue of Ubiquitous Computing. Pervasive computing systems run on an extremely dynamic computing environment, so auto-sensing objects in the environment and its changing status information are important features of ubiquitous computing.
Ubiquitous computing systems from their own terms consists of the following main actions:

- recording the user's operation and the context information of each operation;
- judging the user's behaviour by analysing of the history and present context information;
- determining the context environment around users and the devices;
- predicting the user's needs and perform the appropriate pre-operation.

The context information is the key point of Ubicomp. The questions "what context is" and "how it can be used" must be discussed first of all, in order to take all advantages of the context information. An understanding of context will enable application designers to choose what context to use in their

applications. An understanding of how context can be used will help application designers determine what context-aware behaviors to support in their applications.

2.4.1 Definition of "Context"

The word "context" is defined as the circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood in Oxford Dictionaries.[5] The use of the word "context" tends to be vague because everything in the world happens in a certain context.
The term has been used in many ways in different areas of computer science, such as "contextsensitive help", "contextual search", "multitasking context switch", "psychological contextual perception", and so on. We only focus on the context used by applications in ubiquitous computing.
Context is usually referred to the attributes in a continually dynamic computing environment in which various aspects like the user's location, social situation and interaction with other resources are alternating constantly. Context is any information that can be used to characterise the situation of an entity.
The information in the physical, social and digital environments creates a context for the interaction between humans and computational services.

2.4.2 Context Categories

A categorization of context types will help designers uncover the most likely pieces of context that will be useful in their applications. Previous definitions of context seed our development of context types.
Schilit and his partners list the important aspects of context as[6]:

- where you are
- who you are with
- what resources are nearby

N. S. Ryan and J. Pascoe and D. R. Morse suggest context types of[7]:

- location
- environment
- identity
- time

Krogstie's categories are more fine grained[8]:

- Spacio-temporal context
- Environmental context
- Personal context
- Task context
- Social context
- Information context

There are many types of context information and their different properties lead to different ways to express and model them. To our knowledge, all the current systems use their own way to model the contextual information. It is impossible to exchange context information between them, or to notify the applications on one system about context changes based on the context sensed by another system. There are certain types of context that are, in practice, more important than others. These are location, identity, activity and time.[9]

2.5 Summary

The above discussed two characteristics can be mapped into the three main constituents of the dynamical environment.

2.5.1 Sensor-Based Contexts in physical world

Most research of modeling context focuses on location information only.[10] As for the navigation, it can guide the user by determining the present position throughout the satellite information and calculating the suitable way to the destination depending on the information from selected maps.
Besides location property, time, width, temperature are also easy to be measured. All the physical features are used to describe the environment around the object and status of the user or the device.

2.5.2 User Contexts in social world

Context-aware applications look at the who's, where's, when's and what's (that is, what the user is doing) of entities and use this information to determine why the situation is occurring. [9]
For example, in a context-aware online shop, a user search some interesting products (maybe: toothbrush), and the system will display the information relevant to the this product (maybe: kind, prices etc.) or some similar products

such as electric toothbrush, or some related products such as toothpaste on the computer. In this situation, the designer has encoded the understanding that when a user approaches a particular product (the incoming context), it means that the user is interested in the product (the reason) and the application should display some relevant information (the action).
In the above example, the business process should be modeled as well as the production. The user's history favours will also help to make the system better.

2.5.3 Complex Distributed Computing Contexts in digital world

The most common context information from digital world is Topology. And digital devices are complex-distributed everywhere around people. For example, the information is parallel independently collected from different sensors or maybe from the same one, in order to verify the consistency and correctness. The structure of related devices must be integrated in the system design process.

3 Modeling Approaches

3.1 Model-Driven Development

There are several development processes known in many different shapes. Most of the traditional software development approaches suggest four core phases: analysis, design, implementation and deployment. Model-Driven Development (MDD), a new scheme to develop a system through modeling, is not an alternative to known software development process, but can be used in the same phases as traditional development. However, a high formalization is required. With MDD a software system is specified through models on a very abstract level.
A model is used for representing an object, an environment, a process or other elements in a logical and objective way. All the models are reflections of reality. In the field of software engineering, a model is an abstracted reflection of a system including the definition of a set of rules, which can interpret the different meanings of its components.[12]
A modeling process can be divided into four levels as shown in Figure 2, called respectively as M0 (Instances), M1 (Model), M2 (Meta Model) and M3 (Meta meta model). The modeling of context-aware systems will be achieved through the definition of models on M1 level which conform to meta-models on M2 level.
Most development work takes part on M1 layer , as on this layer concrete

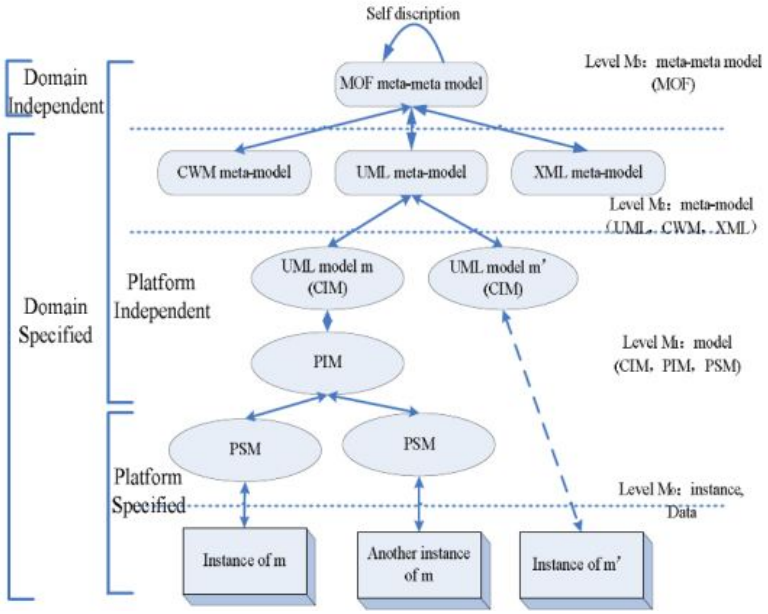


Fig. 2. MDA model hierarchy[11]

systems are specified. Model-Driven Architecture(MDA) defines system functionality on this layer, which was launched by the Object Management Group(OMG) and has been the most known and most supported instance of MDD at present. MDA provides concrete measures to apply MDD to current software development processes.

In different phases of the traditional software development process, several artifacts are specified by different languages, such as natural language as well as source code, which must be manually transferred to the next phase. To overcome the problem, each phase with MDA results in a model as artifact and therefore of a higher specification and formalization degree. Changing the high level specification from diagrams and text to formal models that support a stepwise transformation to code is the core idea of the MDA.

As the table 2 shows, MDA uses four types of models which differ in their level of abstraction.

These four models can be specified by modeling languages. They start on an abstract level and then transformed to more concrete models, finally transformation generates source code. Model represents implementation, changes to model are propagated to source code by transformations.

Modeling languages aim to describe the model in a formal or a semi-formal

Development Phases	**Artifacts of traditional architecture**	**Artifacts of MDA**
Analysis	Informal Text *natural language*	Computation Independent Model (CIM)
Design	Diagrams and Text *modeling language* *pseudo code...*	Platform Independent Model (PIM)
Implementation	Diagrams and Text *programming language*	Platform Specific Model (PSM)
Deployment	Code *native code*	Platform Specific Implementation (PSI)

Table 2. Artifacts in different Phases of Traditional architecture and MDA

way, which can be textual or graphical. Unified Modeling Language (UML) is now a standardized modeling language and used throughout the software development life cycle.
Tools have been defined to implement the MDA, such as:[13]

1. Creation Tool
2. Analysis Tool
3. Transformation Tool
4. Composition Tool
5. Test Tool
6. Simulation Tool

Kolos-Mazuryk compared different ways to model context in his Ph.D. thesis [4]. He suggests that a context model should consist of Context-attributes, the specification how these attributes are changed, the reaction of the service to changes and activities which cause attribute changes. Moreover the actors which perform the activities and furthermore interdependencies between context attributes.

3.2 Simulation Models

Simulation improves the understanding of a system without having to actually handle it, either because it is not yet defined or not available or because it

cannot be manipulated directly because of cost, time, resources or risk.
The modeling stage is the most critical phase of the simulation. To get a good modeling several issues must be addressed. It is necessary to :[12]

- Analyze the problem to solve and set goals matching the specifications,
- Define the inputs and outputs of the system,
- Identify the interactions between the elements and build a conceptual model,
- Identify the dynamics of the system i.e., the system behavior over time in its environment (super-system),
- Study the most relevant elements of the system.

Simulation model can represent the ubicomp system, but it can only calculate the next state of the system based on the present and past state.

3.2.1 CD++

CD++ Builder (see in Figure 3) is an Eclipse plugin for Discrete-Event modeling and simulation, based on DEVS-formalism (abbreviating Discrete Event System Specification) which can simulate a large Wireless Sensor Network (WSN) by implementing the Topology Control Algorithm as presented in.[14]

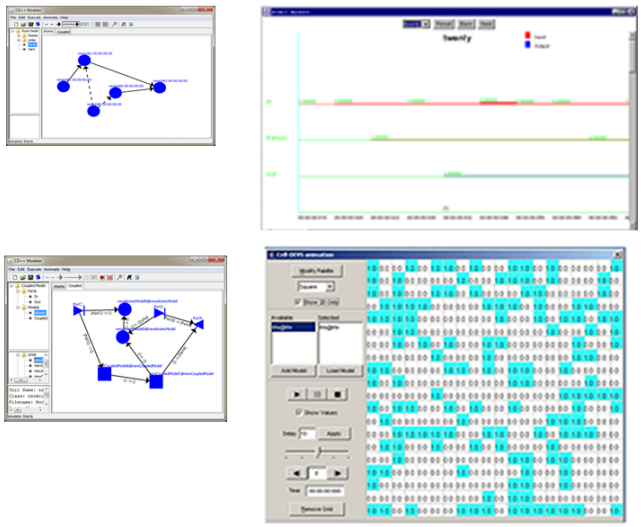


Fig. 3. Development Environment of CD++

DEVS is a sound formalism for modeling and analysis of discrete event systems by describing generic dynamic systems in a hierarchical and modular way. Cell-DEVS is an extension to the DEVS formalism[15], which intended to model physical systems and describe the complex distributed networks. It includes the definition of cellular models, which is represented as a cell space, where each cell is represented as an atomic DEVS model. An Atomic Cell-DEVS can be defined as follows:[14]

$$TDC =< X,\ Y,\ I,\ S,\ \theta,\ N,\ d,\ \delta int,\ \delta ext,\ \tau,\ \lambda,\ D > \quad (1)$$

with:

X:	the set of external input events
Y:	the set of external output events
I:	the definition of the model's modular interface
S:	the set of possible states for a given cell
θ:	the definition of the cell's state variables
N:	the set of values for the input events
d:	the delay of the cell
δ**int**:	the internal transition function
δ**ext**:	the external transition function
τ:	the local computing function
λ:	the output function, and
D:	the duration function.

The element in S could be a context attribute from a sensor in physical environment, which contains contextual information.
Once each cell is defined, they can form a coupled model:[14]

$$GCC =< Xlist,\ Ylist,\ I,\ X,\ Y,\ n,\ \ t1,\ ...,\ tn,\ N,\ C,\ B,\ Z, select > \quad (2)$$

Where besides I, X, Y, those are defined as the same as the atomic model:

Xlist:	the input coupling list
Ylist:	the output coupling list
n:	the dimension of the cell space
$\{t1,\ ...,\ tn\}$:	the number of cells in each of the dimensions
N:	the neighbourhood set

C:	the cell space
B:	the set of border cells
Z:	the translation function
select:	the tie-breaking function

The function of the distributed computing is defined using a set of rules with the form:

$$VALUE \ \ DELAY \ \ \{PRECONDITION\}$$

which means: if the $PRECONDITION$ is satisfied, the state of the current cell will change to the designated $VALUE$, and after $DELAY$ it will be transmitted to other components.
This specification defines a coupled model composed of an array of atomic cells. Each of them is connected to its neighborhood.[15] With the array in neighbourhood, the connection relationship among the cells can be deduced, the structure of the WSN is thus analysable and the complex-distributed property can be modeled.
Cell-DEVS is only a formalism, to implement is a tool required. CD++ is a toolkit to realize this simulation model[14], that was built using a standard version of C++, which allowed is to provide different versions running in several platforms, such as windows and Linux, without additional cost. But it can run either in standalone mode or in parallel mode.[16]

3.2.2 COOJA

Some simulators can only simulate a single level of a system at once. This makes system development and evolution difficult since developers cannot use the same simulator. For example, the above discussed CD++ with DEVS can be used at operating system level or at the networking level, but it's impossible to use the same Simulator at the both levels.

COOJA, a novel simulator for the Contiki operating system, is developed to simulate at the following three levels of system shown in Figure 4:

1. Networking Level
2. Operatin System Level
3. Machine Code Instruction Set Level

COOJA has advantages in terms of efficiency and memory usage[17], it simulates wireless sensor networks of Contiki nodes. Contiki, which is said as

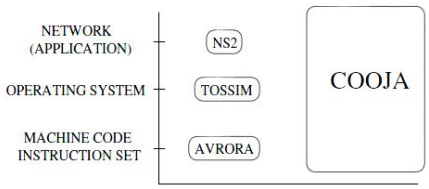


Fig. 4. COOJA can simultaneously simulate at several levels.[17]

"The Open Source OS for the Internet of Things"[18], is a small and easily portable multi-tasking operating system. It is designed to suit a range of memory-constrained network systems. With Instant Contiki and Cooja (see in Figure 5), Contiki is more easy to install and get started with and can be deployed at different levels with suitable configuration.

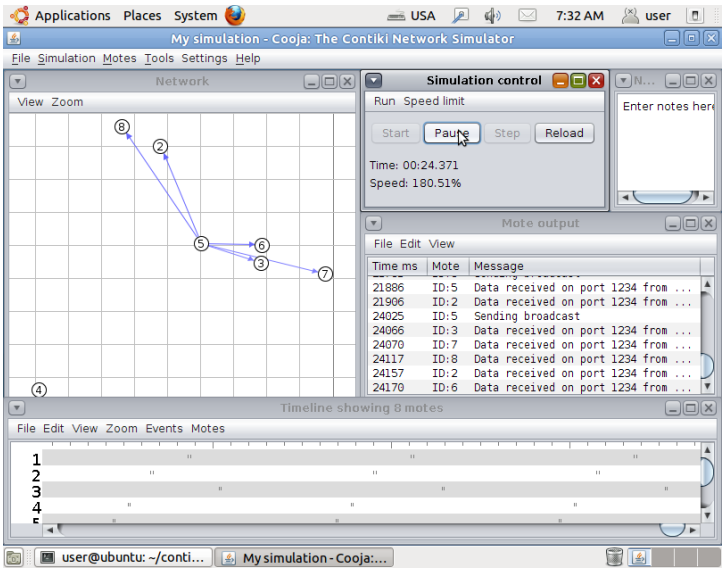


Fig. 5. Instant Contiki with Cooja[18]

There are three types of theses nodes: emulated nodes, which emulate the entire hardware, native Cooja nodes, where the Contiki code for the node is compiled for and executed on the simulation host, or Java nodes, whose behavior must be reimplemented as a Java class. A single Cooja simulation

may contain a radio model any of the three classes. Emulated nodes can also be used to include non-Contiki nodes in a simulated network.[17]
The cross-level sensor network simulator intends to make nodes from each level co-exist and interact in the same simulation. In this way, COOJA can also simulate the contextual information from the sensor node. The simulation on networking level represents the complex distribution and gather the data from sensor on Machine code instruction set level. The cross-level feature makes COOJA particularly suitable for pervasive systems.

3.3 Model-Checking

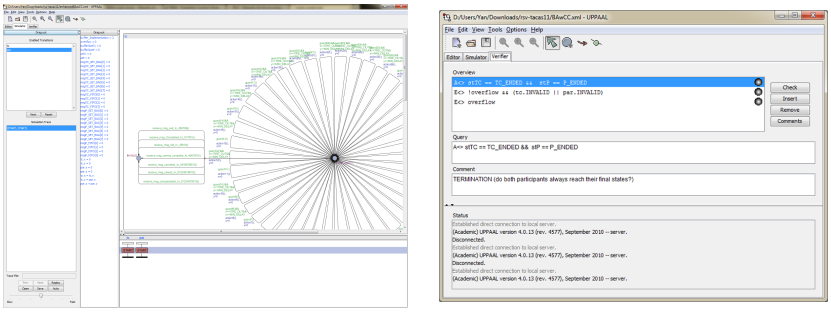
Model checking is an important automated verification technology, and how to use the technology to ensure security and reliability of software has become a hot research topic in recent years. Software model checking methods and their realized prototype systems have been described in this part.
With a simulator, only possible behaviours can be explored, that is unable to reveal the worst-case as well as the best-case scenarios. To check that some requirements are guaranteed by all possible behaviours of a network, the model checking is required to complement simulations. The purpose of model checkers is to check automatically, whether a given model satisfy a given specification.
Distributed systems have proven to be hard to understand and design due to their complexity and non-deterministic nature. One of the promising solutions to this problem is the use of formal verification techniques such as model checking technique. Ubicomp systems, known as a complex distributed system, should take advantage of Model-checking in the development process.

3.3.1 UPPAAL

From the result of [19], compared with traditional simulation model, "the wireless channel model should be improved in UPPAAL - especially for networks with multi-hop communication".

UPPAAL, shown in Fig.6, is a tool box based on the theory of timed automata, which provides a modelling language, a simulator and a model checker.[21] The included model checker is designed to check for invariant and reachability properties based on an interpretation using a finite-state symbolic semantics of networks, in particular whether certain combinations of

[3] The Example is to model and check Web Services Business Activity Protocol from the research of Aalborg University[20]



(a) Simulator of UPPAAL (b) Model-Checker of UPPAAL

Fig. 6. UPPAAL 4.0 in Windows [3]

control-nodes and constraints on clocks and integer variables are reachable from an initial configuration. In UPPAAL, processes are modelled as communicating timed automata and can be animated to explore specific paths based on derived scenarios.
Timed automata can be described in Figure 7 as example, where X is defined as CLOCK.

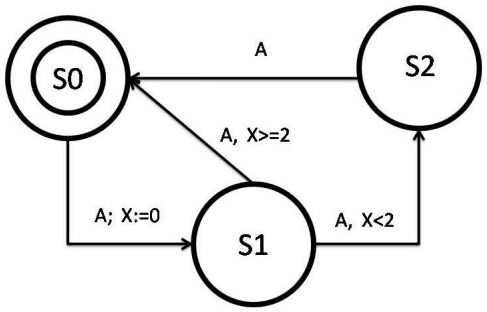


Fig. 7. Timed Automata

Using Clock can make the distributed computing system simultaneous and synchronous, that is useful to model ubicomp systems. UPPAAL 4.0 is able to check the reachability from an initial configuration.[21] Checking reachability of ubicomp system can make sure whether the system can satisfy the requirements.

3.3.2 PRISM

PRISM, as probabilistic model checker, is a tool to model and analyse the systems that exhibit random or probabilistic behaviour.[22] Models are formulated in the PRISM language and can be further analysed.
Probabilistic models can be classified into by modelling of time and the presence of non-determinism. In PRISM six main probabilistic models can be built and analysed[23].

1. discrete-time Markov chains (DTMCs)
2. continuous-time Markov chains (CTMCs)
3. Markov decision processes (MDPs)
4. probabilistic automata (PAs)
5. probabilistic timed automata (PTAs)
6. priced probabilistic timed automata (PPTAs)

Exact verification of probabilistic properties is the core of PRISM. Unlike Prism, UPPAAL focused on finite timed automata, the new version UPPAAL-PRO can support PTAs now, but is only able to analyse maximum probabilistic reachability problem.[23]

3.4 Execution-Based Model-Checking

Traditional model checker based on the reachability problem and are used before deploying to verify, whether a designed model can satisfy the required specifications. In contrast the execution-based model checking approach is enumerative verification, which can implement enumerative state space exploration.[24]

3.4.1 T-Check

T-Check building upon TOSSIM[4] employs both explicit state model checking and random walks to make these sorts of problems reveal themselves prior to deployment.[25] AS the result shown in [26], it can find more unknown bugs in sensor network applications running on TinyOS[5] than the early checkers.
The errors found by T-check can be classified into safety and liveness errors. A great quantity of safety properties are inherit from Safe TinyOS[26], those are compiler-generated and because of complex distributed computing can't be

[4] TOSSIM is a simulator for TinyOS Networks.
[5] TinyOS is an embedded operating system for programming wireless sensor network device,which is programmed in nesC.

ignored. Developers should defined the other safety properties and all the liveness properties.

3.5 Other Tools

3.5.1 TEPAWSN

There are some tools which do not fall into either simulation or verification categories. Unlike UPPAAL, this is also an integrated tool environment for modeling, validation and verification. But its chief distinguishing feature is Model-checking.

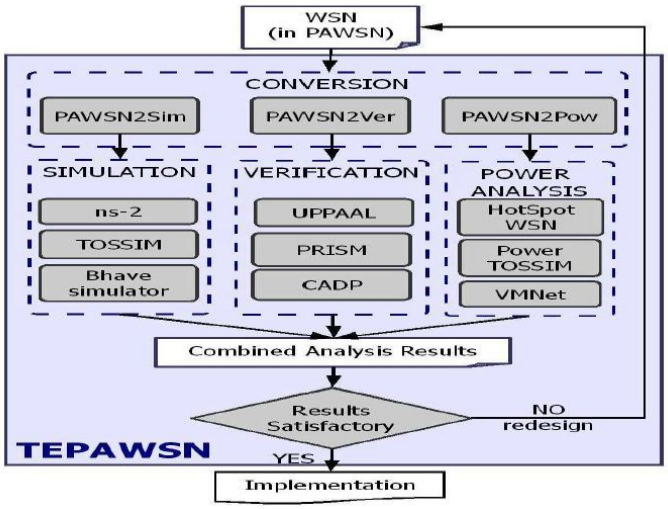


Fig. 8. TEPAWSN Architecture[27]

In contrast, TEPAWSN, which is a tool environment for WSNs simulation and verification, is famous for its conversion tools, such as PAWSN2Sim, PAWSN2Ver and PAWSN2Pow. They are defined to transform the WSNs described in PAWSN into the corresponding simulator, verification and other analysis component. The formal language PAWSN (Process Algebra for WSNs) is able to describe the timed, non-deterministic and/or probabilistic behaviour of WSNs including the power issues[28].
System development using TEPAWSN consists of several components, namely: visualization, simulation, verification and implementation. The procedure is

clearly visible from the reference architecture depicted in Figure 8. Three groups of conversion tools are defined for the different purposes[27].

4 Case Study

4.1 Use Case

In order to clarify the advantages and disadvantages of the tools, a simple use case will be defined. The simple example is an automatic lighting control system at home, which holds the dynamic characteristics of ubicomp systems. Only two types of sensors will be used, an occupancy sensor and a photosensitive sensor. Time-clocks are added into the computing logic. And as supposed, there is only a the lamp in each room, which can be either on or off. The sub-controller in each room can collect the information from the sensor and lamp located in this room, and it is the sub-controller's task to turn on or turn off the lamp and the rules are shown in Fig.9. The signal "occupied" comes from the occupancy sensor and indicates whether the room is occupied or not. The other signal "insufficient" comes from the photosensitive sensor and indicates whether the light in the room is sufficient or insufficient. The states of lamp are: off and on. The sub-controller will turn on the lamp, only if the both conditions, namely "occupied" and "insufficient" are met. By contrast, only one dissatisfaction with the both conditions will cause the lamp to turn off. There is a master-controller in the whole house to analyse the information from the sub-controllers.

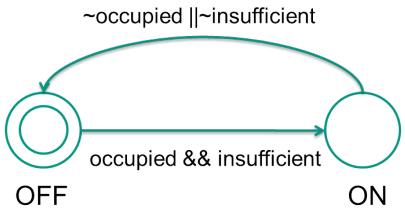


Fig. 9. State Machine of the lighting controlling rules in single room

4.2 CD++

In order to simulate the system, sensors will be described as atomic Cell-DEVS.

4.2.1 Occupancy sensor

$$OCC_{sensor} =< X,\ N,\ Y,\ S,\ d,\ \delta ext,\ \lambda,\ D > \tag{3}$$

where

- $X = \{?occupied\}$
- $N = \{true, false\}$
- $Y = \{!send(status)\}$
- $S = \{true, false\}$
- $d = 60\ sec.$
- $\delta ext(true) = \{if(?Occupied = false)S := false\}$
- $\delta ext(false) = \{if(?Occupied = true)S := true\}$
- $\lambda(true) =!send(true)$
- $\lambda(false) =!send(false)$

4.2.2 Occupancy sensor

$$PHO_{sensor} =< X,\ N,\ Y,\ S,\ d,\ \delta ext,\ \lambda,\ D > \tag{4}$$

- $X = \{?lighting\}$
- $N = \{sufficient, insufficient\}$
- $Y = \{!send(status)\}$
- $S = \{sufficient, insufficient\}$
- $d = 60\ sec.$
- $\delta ext(sufficient) = \{if(?Occupied = insufficient)S := insufficient\}$
- $\delta ext(insufficient) = \{if(?Occupied = sufficient)S := sufficient\}$
- $\lambda(insufficient) =!send(insufficient)$
- $\lambda(sufficient) =!send(sufficient)$

4.2.3 Sub-controller

The sub-controller can be defined as a coupled model, as in CD++ could use rules to describe its distributed computing rules.

And all the sub-controllers can be coupled as a model for the whole system. With the defined models, predictions will be made about the behaviour of the system in certain hypothetical situations.

type:cell neighbors: $OCC_{sensor}, PHO_{sensor}$			
RULES	VALUE	DALAY	$\{PRECONDITION\}$
rule 1	turn on and !send(on)	60secs	$\{Sub-Controller == off$ **AND** $OCC_{sensor} == true$ **AND** $PHO_{sensor} == insufficient\}$
rule 2	turn off and !send(off)	60secs	$\{Sub-Controller == on$ **AND** $(OCC_{sensor} == false$ **OR** $PHO_{sensor} == sufficient)\}$

Table 3. some rules of the use case

4.3 COOJA

With COOJA, the sub-controller can be defined as a contiki node. At operating system level, it can monitor the environment of the room and make decision, if turn on or off the lamp. At networking level, it can communicate with each other, so the whole system can for example calculate the number of the lamps burning. At hardware level, it can receive the environment status as contextual information from the lamp and sensors. As it can cross level simulate the system, it's much easier to deploy it.

4.4 UPPAAL

With UPPAAL the reachability of this system can be discussed. The models of the system in the Simulator of UPPAAL can be divided into occupancy sensor, photosensitive sensor, sub-controller and master-controller. The controlling rules can be checked by being written in specification.

4.4.1 Global declarations

```
1 clock x;
2 const int N = 5;          //#rooms
3 typedef int[0,N-1] room-id;
4 bool light[N];
5 const int Max=3;
6 int sum=0;
7 chan occupied[N], unoccupied[N], sufficient[N],
8       insufficient[N];
9 \textbf{system:} Occupancy sensor, Photosensitive sensor,
```

```
10          Controller , Overhead;
```

4.4.2 Master-controller

Overhead is defined to collect the information of the whole system and make sure the sum of the opened light is under the range. (see Fig. 10)

```
11 void initial() {
12 for (i: int[0,N-1])
13 light[i]=false;
14 }
```

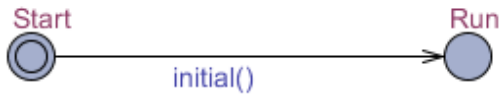


Fig. 10. Master-Controller

4.4.3 Sub-Controller

Controller is equipped in each room and supposed to open the light when the preconditions are satisfied. (See in Fig.11)

```
15 Parameter: const room-id id
16
17 void turnON(int i){
18 light[i]=true;
19 }
20
21 void turnOFF(int i){
22 light[i]=false;
23 }
```

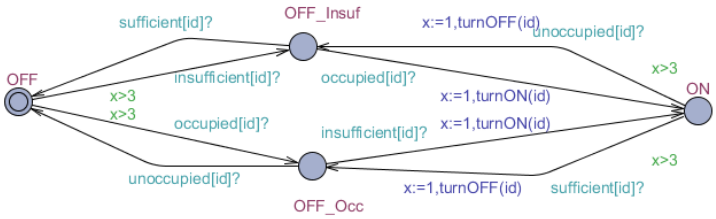


Fig. 11. Sub-Controller

4.4.4 Occupancy sensor

Occupancy sensor represents context information of the occupancy. (See in Fig.4.4.4)

```
24 Parameter: const room-id id
25
26 local declaration: int b;
```

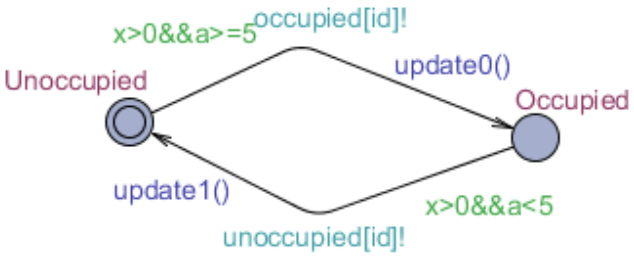


Fig. 12. Occupancy Sensor

4.4.5 Photosensitive sensor

Photosensitive sensor represents context information of the sufficiency of the lighting.4.4.5

```
Parameter: const room-id id

local declaration: int a;
```

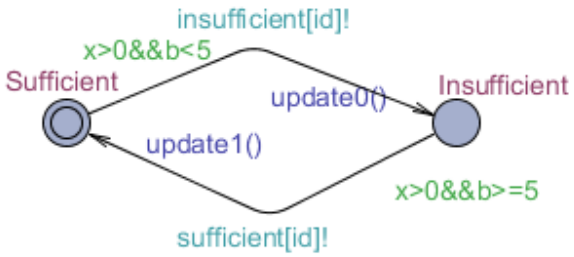


Fig. 13. Photosensitive Sensor

The simulator of UPPAAL can combine the needed model into a system as shown in Fig.14.

4.4.6 Specification

The following specifications are written to check, whether the system can meet the requirements.

```
E<>    Photosensitive-sensor(1).Sufficient &&
       Controller(1).ON
E<>    Photosensitive-sensor(1).Sufficient &&
       Controller(1).OFF-Occ
A[~]   forall(i:room-id)
       Photosensitive-sensor(i).Sufficient imply
       Controller(i).OFF || Controller(i).OFF-Insuf ||
       Controller(i).OFF-Occ
A[~]   forall(i:room-id) Occupancy-sensor(i).Unoccupied
```

```
39          imply Controller(i).OFF || Controller(i).OFF-Insuf
40          || Controller(i).OFF-Occ
41  A[~]    forall(i:room-id) Occupancy-sensor(i).Occupied &&
42          Photosensitive-sensor(i).Insufficient imply
43          Controller(i).ON
44  A[~]    not deadlock
```

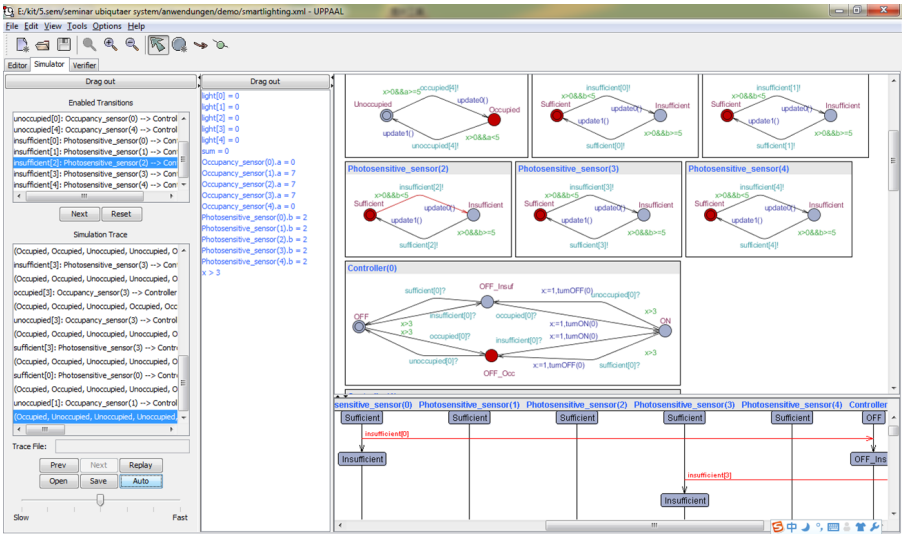


Fig. 14. The user interface of simulator in UPPAAL

4.5 TEPAWSN with PRISM

Because a simple system is under consideration, the features of PRISM cannot be advanced. But if we use a User-Alarm instead of an occupancy sensor to model the system, we should use PRISM, because the probability of a person entering the room.

4.6 T-Checker

T-checker can find bugs before implementing. The system should be simulated in TISSOM and can use T-check to test its safety and liveness.

5 Conclusion

In this paper, the ubicomp system is discussed at first and then some tools are introduced, which can be used during modeling ubicomp systems. They are compared using a use case in Section 4. In summary, the profile of those tools are listed in Table 4.

	Functions	GUI	Operating System	Feature	Purpose
CD++	Simulator	supported	Windows (Eclipse plugin)	Based on DEVS/Cell-DEVS Formalism	simulate the result following the given specifications
COOJA	Simulator	supported	Linux (Contiki)	Cross-level	simultaneous simulation crossing three levels
UPPAAL	Simulator, Model-Checker	supported in simulation	Windows, Linux, MacOS	Timed Automata	verify program properties, e.g. reachability, concurrency, invariants etc.
PRISM	Simulator, Model-Checker	supported in simulation	Windows, Linux, MacOS	PTA etc.	check the probabilistic properties
T-Check	Execution-Based Model-Checker	unsupported[a]	Linux (TinyOS)	Explicit state model checking and random walks	check safety and liveness properties
TEPAWSN	Tool Environment	supported	Windows, Linux, MacOS [b]	Translator among different components	Simulation, verification and implementation of WSNs

Table 4. Comparison of the modeling tools

[a] simulation is based on TOSSIM with GUI
[b] depending on the chosen components

As said in Section 2.5, the context information come from physical, social and digital worlds.

The physical contextual information are the most easy part to represent, those can be a set of inputs of any model. Like in use case of section 4, the "sufficient" and "insufficient" is a result from a photosensitive sensor. The photosensitive sensor may just give a value about the lighting status in room, but with a given threshold, the value can be transformed just like a boolean type.

In the digital world, complex distribution is one of the most important characteristics of ubiquitous computing systems. Simulators can represent this character better than model checkers. The complex distributed computing structure of ubicomp systems could be described by CD++ using the rules of each coupled cell-model described and by COOJA using the contiki-node at networking level. Model checkers can only verify program properties. Execution-based model checkers, e.g. T-checker, because of its existent specifications, are able to find the bugs those may happen during the execution of a given distributed computing system model before implementation.

The social context can always be expressed as process diagrams and model-checking is most suitable to confirm, whether requirements of the system are satisfied. UPPAAL based on timed automata focuses on the reachability. In contrast, PRISM can explore all the possible states based on Markov chains and probabilistic timed automata.

In conclusion, the simulator should be used to design a model and the possible states can be calculated . After that, a model-checker can be used to confirm, whether the required specification is satisfied and the reachability of the designed model. Execution-based model-checking will be added to find potential running bugs before implementing.

References

1. Weiser, M.: Some computer science issues in ubiquitous computing. Commun. ACM **36** (1993) 75–84
2. Weiser, M., Brown, J.S.: The coming age of calm technology. In Denning, P.J., Metcalfe, R.M., eds.: Beyond calculation. Copernicus, New York, NY, USA (1996) 75–85
3. Haddad, W.M., Chellaboina, V.: Nonlinear Dynamical Systems and Control. Princeton University Press (2008)
4. L. Kolos-Mazuryk, P. van Eck, R.W.: A survey of requirements engineering methods for pervasive services. Freeband A-MUSE deliverable **D5.7a TI/RS/2006/018** (2006)
5. Dictionaries, O.: Context. `http://oxforddictionaries.com/definition/english/context?q=context` (2013)
6. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications. WMCSA '94, Washington, DC, USA, IEEE Computer Society (1994) 85–90
7. Ryan, N.S., Pascoe, J., Morse, D.R.: Enhanced reality fieldwork: the context-aware archaeological assistant. In Gaffney, V., van Leusen, M., Exxon, S., eds.: Computer Applications in Archaeology 1997. British Archaeological Reports, Oxford, Tempus Reparatum (1998)
8. Kolos-Mazuryk, L., Poulisse, G.J., van Eck, P.: Requirements engineering for pervasive services. In SecondWorkshop on Building Software for Pervasive Computing. Position Papers., San Diego, California, USA. No publisher, (2005) S. 18 – 22.
9. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. HUC '99, London, UK, UK, Springer-Verlag (1999) 304–307
10. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA (2000)
11. Kim, J.H.: Advanced methods, techniques, and applications in modeling and simulation. In: Asia Simulation Conference 2011, Seoul, Korea,. (2011)
12. Amara Touil, Makhlouf Benkerrou, J.V.F.L., Parc, P.L.: Modeling, analysis and simulation of ubiquitous systems using a mde approach. International Journal On Advances in Intelligent Systems **4, 3&4** (2011) 147–157
13. Wikipedia: Model-driven architecture. `http://en.wikipedia.org/wiki/Model-driven_architecture` (2013)
14. Qela, B., Wainer, G., Mouftah, H.: Simulation of large wireless sensor networks using cell-devs. In: Winter Simulation Conference. WSC '09, Winter Simulation Conference (2009) 3189–3200
15. Ameghino, J., Wainer, G.: Application of the cell-devs paradigm using n-cd++. In Proceedings of the 32nd SCS Summer Computer Simulation Conference, Vancouver, Canada. (2000)

16. Wainer, G.A., Wenhong Chen, Juan Ignacio Cidre, E.G.S.L.A.M.A.T.: Cd++: A tool for and devs and cell-devs modelling and simulation (2004)
17. Oesterlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level sensor network simulation with cooja. In: LCN'06. (2006) 641–648
18. OS, C.: Contiki. `http://www.contiki-os.org/` (2013)
19. Tschirner, S., Xuedong, L., Yi, W.: Model-based validation of qos properties of biomedical sensor networks. In: In Proceedings of the International Conference on Embedded Software (EMSOFT 2008, Accepted (2008)
20. A.P. Ravn, J. Srba, S.V.: Modelling and verication of web services business activity protocol. In: 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2010. (2010)
21. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer **Volume 1** (1997) 134–152
22. Checker, P.M.: Prism model checker. `http://www.prismmodelchecker.org/` (2013)
23. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., Qadeer, S., eds.: Proc. 23rd International Conference on Computer Aided Verification (CAV'11). Volume 6806 of LNCS., Springer (2011) 585–591
24. Jhala, R., Majumdar, R.: Software model checking. ACM Comput. Surv. **41** (2009) 21:1–21:54
25. UTAH, C.: T-check. `http://www.cs.utah.edu/~peterlee/tcheck/` (2013)
26. Li, P., Regehr, J.: T-check: bug finding for sensor networks. In: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks. IPSN '10, New York, NY, USA, ACM (2010) 174–185
27. K.L. Man, T. Krilavicius, T.V., Leung, H.: Tepawsn: A formal analysis tool for wireless sensor networks. International Journal of Research and Reviews in Computer Science (IJRRCS) **Vol. 1, No. 1** (2010) 24 – 16
28. Man, K., Vallee, T., Leung, H., Mercaldi, M., van der Wulp, J., Donno, M., Pastrnak, M.: Tepawsn - a tool environment for wireless sensor networks. In: Industrial Electronics and Applications, 2009. ICIEA 2009. 4th IEEE Conference on. (2009) 730 –733

A Survey on Participatory and Personal Air Quality Sensing Systems

Jing Sun and Matthias Budde

TECO / Pervasive Computing Systems
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
jing.sun@student.kit.edu, budde@teco.edu

Abstract. Today's networks and smart phone capabilities allow for interesting social applications involving citizens in environment monitoring. Building a distributed sensing system maintained by many can benefit the community in a way that single institutions can not. This is especially true for air quality sensing. In this paper we will present an overview of latest air quality sensing systems in personal sensing and participatory sensing applications for citizens. We will discuss the current state of these systems, show their limitations and novel approaches.

1 Introduction

Participatory sensing stands for distributed data collection by a group of people with each person contributing their own local observations and sharing this data on a common platform [22]. The number of possible applications is huge, ranging from tracking the spread of diseases to pointing out available parking spots [30]. Some existing participatory sensing projects - like Project Budburst [71] for observing plants - are open to everyone to participate. Often just little knowledge is needed, so a large number of people are potential participants. Special websites like HabitatMap [58] provide a platform for gathering and visualizing crowdsourced data. Users can create themed maps and encourage other users to add markers with information about local points of interest. Markers can be manually added using a computer, as is the case on HabitatMap, or added conveniently by using a smart phone. Today's smart phones already include many sensors like a microphone, accelerometer, camera and GPS which can be used to capture data. Wireless communication allows smart phone apps to easily transmit or fetch data from the Internet and devices in the vicinity. Data can be automatically location-tagged and timestamped, making it simple to add new markers. As more people become connected and as smart phones increase in functionality, participatory sensing applications become more affordable and feasible.

The biggest strength of participatory sensing is involving a large group of users to gather, share, analyze and identify environmental phenomena which would otherwise remain invisible [8]. Citizens can become more aware of their surroundings by taking part in data collecting and observing their own measurements compared to those of other users. *Personal sensing* can utilize the same

measuring devices as participatory sensing applications but puts its emphasis on tracking individual values, e.g. food intake, chronic disease progression or exposure to pollutants. Integrating self-monitoring into everyday life can help users increase life quality and health.

But creating these applications is no easy task. Data should be shared without compromising the user's privacy [9] [11] and the application should be scalable for many users while being user-friendly. In applications that measure specific values e.g. noise levels [29], sensor data accuracy is an issue as novice users can not calibrate sensors without assistance. When sensor readings are aggregated to shape a consistent model, data integrity is also important.

Air quality sensing is a special application inside participatory and personal sensing which focuses on measuring air pollution and weather data (e.g. temperature, humidity). Local authorities in larger cities maintain highly tuned weather and air quality monitoring stations and make sensor readings available for the public online. But official monitoring stations are sparse, e.g. only 37 are deployed in Baden-Württemberg [62] with a total population of around 10.5 million [82]. Air pollution concentrations can vary at a small scale even between neighboring streets and over the course of a day [10]. Thus, official air pollution data is unsuitable for tracking personal exposure levels which depends on routes taken and time spent in places indoors and outdoors. In the ideal participatory sensing scenario, every other citizen has a small connected air quality monitoring station either installed at home or carried around. Many small stations can expand existing monitoring networks, provide a better spatial resolution and give insight to personal exposure.

But air quality sensing is faced with more problems in addition to participatory sensing in general. Unlike other participatory sensing applications which only make use of a smart phone's built-in capabilities, air quality cannot be directly measured without additional sensor hardware, a *sensor platform*. The smart phone can however provide functionality like an interface, data processing, determining the current location and transmitting sensor readings to a server. While official monitoring stations deliver very accurate sensor readings, they require regular maintenance and calibration.

In Section 2 we look at which substances are measured when determining air quality and which techniques can be used. Section 3 begins with a short overview of air quality sensing systems, mostly research systems with little available information. Another twelve systems are described in more detail which consist of research systems (MAQS, CitiSense, EveryAware), commercial systems (AirBase, Dylos, Sensaris, Sensordrone) and open source systems (AirCasting, Air Quality Egg, SmartCitizen, Gasser). We present their capabilities, limits, potential upgrades and highlight promising concepts. Finally, in Section 4 we discuss the current state of air quality sensing systems in respect of participatory sensing applications and present the conclusion in Section 5.

2 Air Quality Sensing

Governmental institutions provide up-to-date air quality data measured by their sensor stations, like the U.S. AirNow website [37] or the Air Quality in Europe website [46]. Both sites use an "Air Quality Index" (AQI) to visualize pollution maps with green to red color overlays and to make measurements easier to understand. However, these indices vary by range and by how they are calculated. AirNow's index ranges from 0 to 500 with values of 100 and higher being unhealthy. This index is calculated separately for Ozone and Particulate Matter $PM_{2.5}$. For comparison, the European index ranges from 0 to 100 and is calculated from measurements of Ozone, PM_{10}, NO_2 (and CO, $PM_{2.5}$, SO_2 where available) combined.

The following list shows substances that are measured and associated with air quality according to the US Environmental Protection Agency [85]. The preferred unit here is *ppm* (parts per million) which can be converted to mg/m^3 (milligrams per cubic meter):

Carbon monoxide (CO) Carbon monoxide is a product of incomplete burning of fuels, most commonly gasoline, gas, coal and wood. This odorless and colorless gas reacts with hemoglobin in blood and affects oxygen transport.

Nitrogen oxides (NO_x) NO_x refers to the sum of NO and NO_2 which are emitted during combustion. Both gases are toxic and can lead to breathing problems and contribute to the creation of Ozone.

Particulate matter (PM_{10}, $PM_{2.5}$) Particulate matter refers to a wide range of substances in the air such as dust (which can be metals, acids, organic chemicals etc.). These particles are mostly emitted by industries and traffic and pose health risks. Due to their small size they can get into the lungs or the bloodstream, causing a variety of problems. The PM notation for particulate matter sized smaller than the given number in μm [1]. Two groups are commonly used: coarse particles PM_{10} and fine particles $PM_{2.5}$.

Sulfur dioxide (SO_2) Sulfur dioxide is mainly emitted by industries and power plants when burning fossil fuels. These highly reactive gases can form other chemicals that can impair the bronchial system.

Volatile organic compounds (VOC) VOCs refer to numerous gaseous organic chemicals with varying health impact. Apart from natural VOCs they are e.g. produced by household products such as paint, cleaning chemicals or pesticides.

Ozone (O_3) Ground level Ozone is formed by chemical reactions between NO_x, VOCs and sunlight, which is why the Ozone level is especially high on hot summer days. Exposure to Ozone which typically occurs outside can damage the lungs.

Other values that are often measured along with the listed pollutants are temperature, humidity and pressure.

[1] This is a widely known but simplified definition for PM. A precise definition is given in [49].

The sensors incorporated in today's mobile phones are not sufficient to directly measure air quality, as special sensors are necessary for gases and particles. While it may be possible to find certain gas or dust sensors in future phones, using a separate sensor platform in conjunction with a mobile phone (e.g. to tag measurements with GPS or to store data) is currently the way to go. The need for a sensor platform causes additional problems that have to be solved. Current sensor platforms that produce accurate sensor data are expensive, bulky and only target expert users, e.g. the DustTrak DRX Aerosol Monitor 8533 (approx. €7000) [84] for measuring particulate matter. Even available portable platforms like the Matter Aerosol DISCmini (approx. €1000) [63] are still too expensive for participatory sensing applications. The mentioned platforms specialize in measuring dust whereas a suitable platform for participatory sensing should be able to produce multiple values of gas/particle concentrations at a lower price. An ideal participatory sensing platform should be inexpensive, portable, energy-efficient and low maintenance. Cheap sensors carry problems like aging, baseline drift, high stabilization time and noise sensitivity that contribute to low quality sensor data [13]. Calibration can improve accuracy but complex procedures that the user has to perform regularly should be avoided in favor of usability.

2.1 Measuring Techniques

To the best of our knowledge, research projects have produced three different approaches to approximate or measure air quality.

The first is by using a model to indirectly estimate pollution exposure, as can be found in the PEIR [24] and the imap [10] projects. A sensor device is not needed since the estimates are calculated only based on mobile phone data (GPS and cell tower location), official data (traffic, weather, population) and map data. This approach is less suited for participatory sensing as no new sensor data is generated but it can be used for personal monitoring of exposure levels.

The second method uses sky observation to infer particulate matter concentration in the air. An Android app is available in Google Play provided by the Air Visibility Monitoring project [27] which does not require additional hardware. When opening the app, the user is prompted to take a picture of the sky and to upload it to a server where it is processed. iSpex [59] is a different project that follows a similar approach. It uses an add-on that covers an iPhone's camera lens to take defined pictures of the sky from which fine dust estimates can be inferred. This project is currently running and a national measurement day was planned for May/June 2013 in the Netherlands.

To mitigate the need for extra hardware, the first two approaches utilize indirect methods with background and context knowledge. However, these approaches are rarely chosen and most projects still work with a sensor device for direct measurement. The rest of this paper will focus on sensor devices and their respective systems that were developed for personal use and participatory sensing.

3 Air Monitoring Systems

In the last decade, numerous projects and studies have created systems and sensor devices for air quality sensing. This chapter covers several systems in regards to their applicability for participatory sensing purposes, namely MAQS, CitiSense, EveryAware, AirBase, Dylos, Sensaris, AirCasting, AirQualityEgg, Smart Citizen, Gasser and Sensordrone. Systems, for which too little information was available to allow detailed evaluation, are listed in the following paragraph:

e-Science [28] Steed et al. from University College London have developed a stand-alone sensor device containing a CO sensor, GPS and logging hardware in 2003. Five of these devices were used in a study and carried by university staff who came to work by bike. A large portion of this work was dedicated to visualizing pollution data on 3D maps.

PM-Air [69] PM-Air is a project by Preemptive Media from 2006 in New York. The lunch box-sized, stand-alone device holds sensors for NO_x, CO and O_3 and automatically uploads data to the PM-Air server. According to the website, a number of these devices were distributed to the public. Users were asked to stay within New York City and pass the device to others users so more people could learn about it. A pollution map using the measured data can be viewed by everyone with the latest data shows readings from September 2006 [70].

N-Smarts [16] N-Smarts is an early study (2008) from UC Berkeley conducted with ten devices in Accra, West Africa for two weeks. Four personal devices were carried by students and the remaining automotive devices were fixed to taxi cabs. This first stand-alone version used for the study was able to measure GPS location, CO, NO_2, SO_2 and O_3. The second "integrated" version had CO, NO_x, temperature sensors and an accelerometer. It was designed to fit in the battery slot of a Nokia N95 phone and communicated via Bluetooth to transmit data to the phone.

The Green Watch [61] The Green Watch is a prototype project from France in 2009 and one of the winners of the "Futur en Seine" event. The basic idea was to equip pedestrians with watch-like devices that could measure O_3 and noise. This measured data would be transferred via Bluetooth to a mobile phone which in turn would upload it to a server. To keep maintenance low, the sensors only reported qualitative values like "good", "correct" and "bad". The concept design was never built and the actual prototype was about the size of a big mobile phone.

Cambridge Mobile Urban Sensing [15] [19] CamMobSens was a three-month study in 2010 carried out in the Cambridge area. Multiple devices were used to capture pollution data: hand-held devices for pedestrians and larger stationary devices, both equipped with sensors for NO_2, NO and CO. GPS was integrated into the sensor device and a Bluetooth connection enabled it to transfer data to a mobile phone running Symbian and a custom Python application. All the data was gathered at a server and made viewable. A second version of the sensor device included GPRS functionality to remove the step of transferring the data to a phone first.

P-Sense [23] Pollution-Sense was developed by Mendez et al. from the University of South Florida in 2011. It appears to be a side project of their G-Sense architecture for participatory sensing. The P-Sense sensor platform can measure temperature, humidity, CO, CO_2 and has two additional sensors that can detect multiple gases. Data is collected and location-tagged by a mobile phone via Bluetooth and uploaded to a server. Using their architecture, measurements can be visualized in real-time on Google Maps, even revealing single data points with timestamps.

OpenSense [1] [67] OpenSense is an ongoing project from ETH Zürich to develop an architecture for a heterogeneous sensor network. Their vision is to combine personal data collectors, sensor nodes on public transport vehicles and governmental stations into one system to enhance accuracy and reduce resource consumption. In Zürich, Switzerland, a handful of sensor nodes are installed on trams measuring O_3, NO_2 and CO. Measurements since April 2012 can be viewed online, along with live pollution maps and tram locations [68]. Portable personal sensor devices have not been integrated into the system yet.

GasMobile [14] From partially the same people behind OpenSense comes a sensor platform called GasMobile. This platform has a sensor for Ozone, a battery and relies on a mobile phone for every remaining functionality (data storage, wireless radios, GPS, etc.). It is plugged into a mobile phone via USB while measuring and an Android app provides the necessary interface for viewing and uploading data. An automatic calibration function using official measurements in the vicinity relieves the user of this difficult task.

Sensor Probes [20] Kuznetsov et al. from Carnegie Mellon University have carried out two user studies in 2010 and 2011. Unlike other projects, sensor devices were supposed to be left at certain places and not carried around by users. The first study used mockup sensor devices to find out how and where they would be placed while the second study used real sensors for dust, exhaust and VOC. They could run up to ten days and would automatically send their sampled data via SMS every five hours [21].

MobileDust [65] This research project from KIT aims at integrating an optical dust sensor into a smartphones. The novel idea of retrofitting a particulate matter sensor as a removable clip-on module to a camera phone does not require any electrical modifications. Instead, the flash and camera of the phone are used as light source and receptor of an optical dust sensor respectively. Two early prototypes have been developed and are currently under testing: an active and a passive version. The general feasibility of the approach has been shown in [6].

netatmo [66] Netatmo is a commercial personal weather station system from France. Their sensor platform consists of two modules: an outdoor module and an indoor module, measuring temperature, humidity, air pressure, CO_2 and sound. The outdoor module sends data via radio to the indoor module which transmits all sensor data via Wi-Fi to the netatmo servers. Data can be accessed by using an iPhone or Android app or by using their REST API. Accessing the data without using netatmo's servers does not seem possible.

CubeSensors [50] [51] CubeSensors is a young Slovenian startup company started in March 2013 selling sensor platform systems for personal indoor environment monitoring. The concept is very similar to the Air Quality Egg presented in Section 3.8. It consists of a wired base station which is connected to the Internet and multiple battery-powered sensor stations (2x2 inches sized "Cubes") which report their readings to the base station wirelessly. This data is then forwarded to the CubeSensors servers where it can be accessed via a web interface or the CubeSensors smart phone app. The idea is to be able to monitor different rooms at the same time and the measured environmental parameters are: temperature, humidity, sound, light, barometric pressure and VOC.

3.1 MAQS: Mobile Air Quality Sensing

MAQS, short for Mobile Air Quality Sensing System is a research project aimed at wearable indoor air quality monitoring from the University of Colorado. Information about MAQS are available on their wiki page [64] which also links to a conference paper [18]. This project has produced multiple prototypes and at least one user study but has shown no public activity over the last year. The Android app seems to be taken down as well while the web interface is still online but requires a login to be accessed.

The MAQS system attempts to monitor indoor air quality by equipping users with wearable sensor platforms called *M-pods* which work in conjunction with the user's mobile phone. This battery-powered sensor platform reads sensor data every 6 seconds. An Android app for MAQS is provided for users which handles live data visualization, room location tracking, receiving sensor readings via Bluetooth and uploading them to MAQS via Wi-Fi. Location tracking is done by using the phone's accelerometer to detect room switches and upon entering a room, the phone performs a Wi-Fi scan and classifies this Wi-Fi fingerprint to the ones saved inside the database to identify a room. Special to MAQS is the possibility to share one M-pod with other users in the vicinity. That means users without own M-pods can connect their mobile phones to a nearby (closer than 2 meters) M-pod to receive sensor readings. Users can also specify which data they want to share publicly using the permission control mechanism.

System:	**MAQS M-pod**
Sensors:	ELT S100 (CO_2), MiCS-5525 (CO), MiCS-5526 (VOC), MiCS-4514 (CO, NO_2), SHT21 (temperature, humidity)
Additional features:	Multiple users can share one M-pod
Dimensions:	undocumented
Price:	undocumented
Availability:	not available
Interface:	Bluetooth, mobile app, web interface
Usage:	Mobile, indoors

Table 1. MAQS M-pod overview

The M-pod sensor platform described in the paper [18] differs from the version described on the wiki, which appears to be a more advanced version. For the first version, only a ELT S100 CO_2 sensor was included to minimize energy consumption and the overall air quality was calculated by using the CO_2 value and applying an air exchange model. The second version doubled the former battery capacity and added another four sensors for CO/NO_2, VOC and temperature/humidity (see Table 1 for details). The exact dimensions of M-pods are undocumented but they are meant to be carried by users and appear to be around the size of a large smart phone.

MAQS tackles a very specific problem by specializing on indoor air monitoring so the focus does not only lie with optimizing M-pods for energy consumption and data quality. Much work has been put into developing a reliable recognition of user movement and identifying rooms. For this purpose, MAQS holds a shared database of room Wi-Fi fingerprints as rooms of which no database entry exists can not be identified. While there is little information about data quality, the MAQS system uses a few interesting techniques that could be also applied to participatory air quality sensing, like local sensor data sharing.

3.2 CitiSense

CitiSense is an ongoing air quality sensing research project by the University of California, San Diego since 2010 [26]. So far, two prototype versions of their sensor platform *SEEnsor node* have been built and multiple users studies conducted involving people around the university in San Diego [25] [2]. No sensor platforms have been released to the public so CitiSense has remained a pure research project.

The system consists of three parts: the SEEnsor node (short of System Energy Efficient sensor node), the user's mobile phone and the CitiSense back-end. Users first pair their mobile phone with the SEEnsor node via Bluetooth and sensor readings are directly streamed to the phone. These readings are tagged with the current GPS location, filtered, analyzed, saved and made viewable immediately for the user to see. Personal readings do not have to be uploaded to the back-end as the mobile phone app itself is able to identify patterns such as prolonged exposure to high values of poisonous gases, depending on certain conditions (e.g. asthma) of the user.

The first version of the SEEnsor node [31] was made from four stackable hardware boards: a control layer, sensor layer, power layer and communication layer. Each layer was responsible for a part of the functionality, e.g. the sensor layer held electrochemical sensors for CO and O_3 from City Technologies. At this stage, the mobile phone app was able to visualize live CO and O_3 readings using EPA Air Quality Index values and color codes for good, moderate, etc. air quality levels. Access control was not implemented and sensor readings were always streamed to the back-end server. As the name System Energy Efficient sensor node implies, the developers have conducted tests to find out how much processing needs to be done on the phone and the back-end to find a trade-off

between processing and communication. They came to the conclusion that local processing is generally more favorable.

The second sensor platform version used a different structure (a single board) and different sensors. Table 2 lists the specifications for this version. The developers stuck to using electrochemical sensors but switched to different brands [57] and added a NO_2 sensor. Attached were an Alphasense NO2-A1 NO_2 sensor, Alphasense CO-AX CO sensor and a Sensoric O33E1 O_3 sensor. Using a 7200mWh Li-ion battery, the SEEnsor node was able to run for 5.35 days when sampling and streaming readings every 5 seconds. New features introduced in this version or later include the personalized web interface and a more powerful mobile phone app. Users could view a more detailed view of current sensor readings (in AQI and ppm), daily peaks and a graph timeline showing readings throughout the day. Another power saving feature was that the phone's GPS would only be turned on when network-based localization would be too inaccurate. The personalized web interface could display history data and Google map overlays showing colored dots representing single readings in their respective locations.

CitiSense stands for a lot more ambitions [26] than what has been built and tested so far. An extensible and adaptive infrastructure is planned which supports citizens, researchers and policymakers alike. That means incorporating existing official monitoring stations to help calibrating SEEnsor nodes, using modeling and learning algorithms to identify how air quality develops, allowing prediction, monitoring personal exposure levels and solving problems concerning sensor data quality assurance, security and privacy. In the first CitiSense paper, the authors mentioned aggregating data and adding noise to an extend that individual privacy is ensured without losing overall data analysis accuracy. Automatically labeling sensor data as indoors or outdoors is another concern to allow users to distinguish and monitor their own homes or offices for which official stations can not provide measurements. Finally, power management issues need to be pursued further to minimize energy consumption in the SEEnsor node and mobile phones.

System:	**CitiSense SEEnsor node**
Sensors:	Alphasense NO2-A1 (NO_2), Alphasense CO-AX (CO), Sensoric O3 3E 1 (O_3), temperature, humidity, barometric pressure [57]
Additional features:	–
Dimensions:	6.7 x 11.0 x 4.0 cm
Price:	approx. $1000 [56]
Availability:	not available
Interface:	Bluetooth, mobile app, web interface
Usage:	Mobile

Table 2. CitiSense SEEnsor node overview

3.3 EveryAware AirProbe

EveryAware [54] is an European Union project started in 2011 for creating sensor platforms and data-processing tools for citizen involvement in participatory sensing. Two major applications have been created by EveryAware: *Widenoise*, a noise pollution monitoring application available in app stores for Apple and Android mobile phones. *AirProbe* is an air quality monitoring system which includes an Android app that works together with their custom sensor platform *SensorBox*.

SensorBox is a stand-alone sensor platform and will start measuring and saving sensor data to a SD card when it is switched on. It includes a GPS shield so it does not need to rely on a mobile phone for location data. Users can connect their mobile phone with the SensorBox via Bluetooth, download all saved sensor readings and forward them to the AirProbe servers. Fetching live readings while remaining paired to the SensorBox is also possible. Using the AirProbe app, data sessions can be annotated with markers including text or photos, share results via social network platforms, view the public map and view graphs of their own sensor readings. The application itself can alert the user to provide annotations, request measurements from point of interests or notify about unusual sensor readings in the vicinity. Shared sensor readings can be viewed on the AirProbe website [55] as aggregated data per session and colored circles indicating overall air quality.

A price tag is not given for the SensorBox but they are aiming for a few hundred Euro per device, of which around €225 are needed for the sensors alone. Whether and when they will make SensorBoxes available to the public is also unknown. The EveryAware report [13] mentions two prototypes that were used in their evaluation.

The SensorBox consists of an external battery pack and two custom-made boards: the sensor board containing all sensors and the control board containing GPS, Bluetooth, SD card and the remaining electronics. Three sensors for CO are used as well as sensors for NO_2, O_3, VOC, exhausts and temperature/humidity. Table 3 provides a detailed list of all included sensors. The developers attempt to solve problems involved with low-cost sensors concerning data quality such as sensitivity to meteorological influences (e.g. sunlight, wind), cross sensitivity and baseline drift by using a multi sensor array and training to calibrate it. The basic idea of training is to use reference sensor station readings to predict readings at the location of the user and calculate calibration variables for the SensorBox from these values. A second approach is having users carry a reference sensor monitor along with a SensorBox and adjust the calibration using the mobile reference readings.

Both communication interfaces, namely the Bluetooth protocol and the transmission of sensor data from the Android app to the AirProbe servers, are documented in the EveryAware report [13]. Data that is sent from the SensorBox via Bluetooth are in a *CSV* (comma separated values) format, containing time, location and sensor readings. A REST web service can receive sensor data in a documented JSON format or send map data to the Android app. Thus, it's

System:	**EveryAware**
Sensors:	Alphasense CO-BF (CO) MiCS-5521 (CO) MiCS-2710 (NO_2) MiCS-5525 (CO) Figaro 2201 (CO, H_2, HC, NO_x) MiCS-2610 (O_3) AS-MLV (VOC) Sensirion SHT21 (temperature, humidity)
Additional features:	logs data to internal memory, GPS
Dimensions:	undocumented
Price:	approx. € 500
Availability:	not available
Interface:	Bluetooth, mobile app, web interface
Usage:	Mobile

Table 3. EveryAware overview

possible to use different sensor platforms to feed the AirProbe database or use a different database to send the SensorBox data to.

3.4 AirBase

AirBase is a small startup company founded in 2010 from Israel, dedicated to building their own air quality sensing platform *CanarIT* and data management backend. They are part of the European Union funded CITI-SENSE project for participatory sensing and have deployed a handful of their sensor platforms around Europe [34]. CanarIT can be purchased directly from the AirBase website [33] by requesting a quote, approximately priced at $800 per device.

The sensor platform is about the size of a lunchbox and includes sensors for NO_2, O_3, VOC, dust, sound, temperature and humidity by default. On their request for quotes page there is the option to swap the O_3 sensor for a different sensor from the following list: Ammonia (NH_3), CO, Hydrogen (H_2), Hydrogen sulfide (H_2S), Methane (CH_4), odor, Perchlorethylene (C_2CI_4) and Sulfur dioxide (SO_2). Very little information is available about which specific sensors they are using, apart from a sparse data sheet [33] describes them as metal oxide sensors. According to the data sheet, sensors are delivered pre-calibrated and any further calibration is handled remotely without intervention from the user.

CanarIT needs a power outlet and can be placed either inside or outside, in which case it requires shelter from direct sunlight and rain. The user has to configure a working Wi-Fi connection or supply a SIM card for the CanarIT to be able to stream sensor readings to the AirBase servers every 20 seconds. From that point on, the platform is almost maintenance-free but it is recommended to vacuum it from time to time to prevent drifting because of accumulated dust.

A working internet connection is mandatory as sensor readings are directly sent to AirBase and the user can only access the data by using the provided web

System:	**AirBase CanarIT**
Sensors:	O_3, NO_2, VOC, PM_{10}/dust, sound, temperature, humidity
Additional features:	–
Dimensions:	18 x 16 x 6.5 cm
Price:	approx. $800
Availability:	AirBase website
Interface:	GSM or Wi-Fi, web dashboard
Usage:	Stationary, outdoor or indoor

Table 4. AirBase CanarIT overview

platform [34]. Using this interface, users can configure public visibility of sensor readings, location and other parameters regarding their CanarIT device. Users can also request to be alerted by SMS or email if sensor readings exceed their configured threshold.

The live sensor network map can be viewed by everyone at `sensors.myairbase.com` [34]. Different colored icons are shown at sensor locations depending on the live readings, indicating good, moderate or poor air quality at one glance. Clicking on an icon brings up a window with detailed sensor readings, time of measurement, graphs for each sensor showing history data and a recommendation whether to keep windows open or closed. It appears that all raw sensor data stored at AirBase is available for download in a CSV format to the public [35] or can be requested using the API.

AirBase has created a commercial, full-fledged air sensing system but there are a few downsides. They do not disclose details about their wide range of sensors or reveal how their remote calibration works for users who are interested in data quality. In addition, users can not refuse to share their data with AirBase and instead configure a different server (e.g. their own) to stream the data to. It should be possible though to use CanarIT in conjunction with a different air quality sensing backend by automatically fetching sensor data from AirBase using the API and forward it somewhere else.

3.5 Dylos

Dylos is a Californian company selling air quality monitors for consumer use since 2006. They specialize on laser particle counters and have released multiple versions of their flagship model, the DC1100. Among users with allergies and air purifiers, the Dylos products are generally well-received for their relatively low price and accuracy [53]. They can be bought directly from the Dylos website [52] or from other online vendors such as Amazon.com.

The DC1100 devices are meant to be used for indoor particle pollution monitoring in your home or office and need a power outlet. They are somewhat bulky in size with an height of 17.78cm compared to most of the other sensor platforms that we cover. But as they are supposed to be kept at one place for longer periods of time, the size is a lesser issue. The user interface is simple: you only need to turn it on and it will start measuring and saving data. Apart

System:	**Dylos Air Quality Monitors DC1100, DC1700**
Sensors:	laser particle counter
Additional features:	logs sensor data to internal memory
Dimensions:	17.78 x 11.43 x 7.62 cm
Price:	DC1100: $200 DC1100 with serial port: $240 DC1100 PRO: $260 DC1100 PRO with serial port: $290 DC1100 PRO for Air Purifier: $320 DC1700 Battery operated: $425
Availability:	Dylos Corporation website
Interface:	LCD display, serial port (optional)
Usage:	Stationary, indoor

Table 5. Dylos Air Quality Monitor overview

from the power button, two additional buttons allow the user to query sensor reading history by minute, hour and day up to 30 days in the past. A two line LCD display with back light shows live readings when the user does not query history data. The first line shows two values for different sized particles: one for particles sized $1\mu m$ to $5\mu m$ and the second for particles larger than $5\mu m$. When showing live readings, this value is a moving average of particle readings in the last 10 seconds. A bar graph in the second line of the display indicates the size of a particle that has just been detected and therefore jumps back and forth very quickly. Concentration values are given in number of particles/100 per cubic foot, so multiplying the value by 100 will give the actual number of particles per cubic foot.

All DC1100 devices use the same laser particle counter with the standard version priced at $200. The Pro version ($260) is calibrated for increased sensitivity to detect particles of sizes $0.5\mu m$ to $2.5\mu m$ and particles above $2.5\mu m$. An upgrade to both versions that includes a serial port to connect to a computer is available for additional $30-$40. The current Dylos software allows live streaming of data from the device, downloading of history data and graphing. Data sessions can be saved to txt files and imported again at a later time for visualization.

A newer version named DC1700 has also been released, featuring improvements such as a battery pack, real time clock to save time-stamped readings and a larger memory. The serial port and increased sensitivity calibration are included by default. Despite being battery powered, it is still designed for indoor use, lasting up to 6 hours per charge.

Maintenance is kept to a minimum by cleaning the fan using canned compressed air. According to Dylos, the main problem of failure is a dying laser but it should last several years.

Dylos advertises their devices to be just as accurate as $4500 particle counters for a significant lower price. They are already used by consumers to determine when to turn on air purifiers and are affordable enough to be used for partici-

patory sensing purposes as well. However, the current software is still lacking in terms of streaming data to an online server or enabling users to write their own by providing an appropriate API. It's not impossible using the current software though, given some manual work by the user to save data sessions and upload them to a participatory sensing system.

3.6 Sensaris

Sensaris [72] is a French company which offers several mobile air quality sensor devices (SensPods) and a data management system (SensDots) to store and view personal sensor data since 2006. According to the website, SensPods have been used in the GreenWatch project (2009) and the Citizen Mobile Environmental Sensing project [48] (2011).

Four different versions which measure different environmental values are currently offered: ECOsense, ECO2sense, ECO3sense and EcoPM. The sensor platforms are mobile, using Lithium ion batteries that can be recharged via mini USB. ECOsense includes uncalibrated MiCS-4514 sensors (CO, NO_x), a Sensirion SHT11 sensor (temperature, humidity) and an unknown microphone (sound). ECO2sense can only measure CO_2 concentrations but this ELT S100 sensor comes pre-calibrated. ECO3sense has a MiCS-2610 sensor (O_3), a Sensirion SHT11 sensor (temperature, humidity) and a SG Lux AG38S for measuring UV radiation. EcoPM is their particulate matter sensor platform using an unspecified Shinyei sensor. Except for EcoPM, every device has built-in GPS to associate measurements with their respective locations. SensPods can only be bought by requesting a quote and estimated prices are unknown.

All SensPods devices can either save sensor data to an embedded 2GB/4GB SD card or transmit them live via Bluetooth to a gateway device, e.g. a mobile phone. For Android phones, an app called MobiSense is provided which can connect to multiple SensPods, show and save live readings or forward them to an arbitrary configured server in a RSS or CSV format. Gateways are necessary to read sensor data from a SensPod and users without compatible mobile phones can use a GuruPlug Bluetooth to Ethernet/3G adapter to forward data.

SensDots is Sensaris' web interface and sensor data management service. The web interface provides graph and map visualizations of an user's own sensor data and the possibility to download historical data. Using this system is not mandatory as sensor data can be streamed to any server. Users are free to write their own mobile applications to fulfill the gateway and forward function.

The applications offered by Sensaris only support personal sensing as there's no functionality to actively share your own sensor data with other people. However, the sensor platforms simply stream data via Bluetooth so they can be integrated into participatory sensing systems using an appropriate mobile phone app.

System:	**Sensaris ECOsense, ECO2sense, ECO3sense, EcoPM**
Sensors:	ECOsense: MiCS-4514 (CO, NO_x), Sensirion SHT11 (temperature, humidity), sound ECO2sense: ELT S100 (CO_2) ECO3sense: MiCS-2610 (O_3), Sensirion SHT11 (temperature, humidity), SG Lux AG38S (UV) EcoPM: Shinyei sensor (PM)
Additional features:	GPS, SD card
Dimensions:	ECOsense: 5 x 8 x 2 cm ECO2sense: 5 x 8 x 3,5 cm ECO3sense: 5 x 8 x 2 cm EcoPM: 7,5 x 9,7 x 4,6 cm
Price:	undocumented
Availability:	Sensaris website
Interface:	Bluetooth, Android app, data stream to arbitrary server
Usage:	Mobile

Table 6. Sensaris SensPods overview

3.7 AirCasting

AirCasting [36] in an open source participatory sensing system and sensor platform by HabitatMap [58], a non-profit environmental health organization in New York. It complements the HabitatMap which is a map information overlay system, that allows users to manually add markers to the map, edit them or create their own environmental maps by assigning certain markers. Markers can include bits of information about matters in the neighborhood of users whether they are good or bad and can be edited by everyone. The AirCasting system also provides an interface to add data to a public map (called CrowdMap) but specializes on automatically captured sensor data.

The AirMonitor sensor platform contains a Figaro TGS 2442 CO sensor, MiCS-2710 NO_2 sensor, HIH-4030 humidity sensor and TMP36 temperature sensor. It's a mobile platform which streams sensor data via Bluetooth to a paired Android phone which provides the GPS location data. Data is saved in sessions so the user has to manually start and stop readings and issue uploads of sessions to AirCasting. The AirCasting app can display live readings, the CrowdMap and a list of captured sessions. Users without an AirMonitor can still contribute by measuring noise levels using their mobile phone's built-in microphone. The app can also forward sensor readings to another Bluetooth device to visualize them, like the AirCasting Luminescent Apparel that glows in different colors depending on the readings.

A detailed tutorial with a list of components and step by step pictured instructions to build an AirMonitor is available on their website. All source code of the AirCasting system and the AirMonitor is made available on Github. There is less emphasis on the sensor platform but on making the AirCasting app and backend as flexible. The app has an API that can receive data from different sensor platforms which do not necessarily measure air quality but arbitrary phe-

System:	**AirCasting**
Sensors:	Figaro TGS 2442 (CO), MiCS-2710 (NO_2), HIH-4030 (humidity), TMP36 (temperature)
Additional features:	-
Dimensions:	7.8 x 8.2 x 7.8 cm
Price:	approx. $180
Availability:	DIY tutorial
Interface:	Bluetooth
Usage:	Mobile

Table 7. Air Casting overview

nomenon. Users can create their own hardware platforms to plug into the system, e.g. the Zephyr HxM heart rate monitor. However, allowing users to add new kinds of data makes the CrowdMap somewhat hard to read with a drop-down of over 50 different to chose from (each with only a few data points). The majority of user data on the CrowdMap are noise level readings.

3.8 Air Quality Egg

Air Quality Egg is a community-driven project developed by enthusiasts from around the world. It first started with two Internet of Things Meetups in Amsterdam led by Casper Coomen (November 2011) and in New York City led by Mark Shepard (December 2011) with roughly 40 people [44]. The project brought together hobbyists, students and professionals for hardware, software and design to create an affordable air quality monitoring system for the masses. Since November 2011, many more meetings have been organized, prototypes built, Air Quality Eggs deployed, tested and developed. Funding was secured by running a Kickstarter fund raising project which lasted for 30 days in March and April 2012 [40]. There was an overwhelming response with over 1000 Eggs and kits sold on Kickstarter alone. The community around the Air Quality Egg still maintains a wiki [41] for documentation and a very active Google Group for discussions and bug reports [42].

What the developers are aiming for is an open, low cost sensor platform that could be used by everyone. It should be able to compare inside and outside measurements and should be modular if the user wants to add additional sensors. For this purpose, a lot of documentation for software, hardware, enclosures and assembly exist [41] [39] to assist interested parties at every skill level to build or set up an Air Quality Egg. Fully assembled Eggs ($185), kits ($140) and sensor shields ($58) can be bought online [45]. While information is available somewhere, the downside is that it's scattered and might be hard to find at times.

The basic concept of the system is that an user owns a Base Egg, which is connected to the internet and one or more Remote Eggs placed inside or outside in the vicinity of the Base Egg. Remote Eggs contain sensors, communicate wirelessly with the Base Egg and only need a power supply to function. The Base

System:	**Air Quality Egg**
Sensors:	MiCS-2710 (NO_2), MICS-5525 (CO), DHT22 (temperature, humidity) Optional sensor shields: MiCS-2610 (O_3), dust sensor[2], MiCS-5521 (VOC)
Additional features:	Wireless communication between Base and Remote Eggs
Dimensions:	5.36 x 5.36 x 8.00 cm
Price:	$185 (assembled)
Availability:	Assembled Air Quality Egg and DIY kits available [45]
Interface:	Ethernet, web dashboard, sharing
Usage:	Stationary, continuous measurements at residence

Table 8. Air Quality Egg overview

Egg is responsible for collecting the data and forwarding it to a data management server which is currently hosted by Xively (formerly known as Cosm or Pachube). A web interface is provided to view this data and users can choose to publish their measurements for the public by registering their Egg and specifying a location. Live sensor readings from active Air Quality Eggs can be viewed on the main website [38] by selecting single Eggs on a map.

Because Air Quality Egg is an ongoing project, multiple versions of the Egg exist and there is the possibility to build your own "Egg" from scratch and integrate your readings into the Xively system. The two versions that were distributed during the Kickstarter campaign are the most common ones. These were the wired Egg with sensors and base communication in one case and the assembled version with Base and Remote Eggs. They share the same built-in sensors however, the MiCS-2710 NO_2 sensor and the MiCS-5525 CO sensor [41]. To compensate for fluctuations and since temperature and humidity affect sensor readings, a respective DHT22 sensor is included. Three additional shields equipped with sensors for O_3, VOC and dust respectively are available to extend the capability of an Egg. This design of the sensor board was derived [40] from an air quality sensor platform called CitizenSensor [47], developed by Joe Saveedra. He worked on this platform since 2009 and unlike the Air Quality Egg, CitizenSensor is a portable system.

What makes the Air Quality Egg special is that all the necessary resources are available for everyone to join in this participatory sensing system. Probably the most important aspect is that it's easy to contribute your own data to the Air Quality Egg network. One of the biggest remaining challenges is to improve data quality, as the sensors currently used are not calibrated and are sensitive to other environmental factors. However, this was an intentional decision and data quality was deemed less important than usability, low price and offering an actual working system. The used sensors are very common online and are often

[2] There are inconsistent statements about which exact sensor is used, either the *Shinyei PPD42NS* or the *Sharp GP2Y1010*.

mentioned in hobbyist forums and websites. So the question remains whether their accuracy can be improved or whether a different solution can be found.

Another problem is the dependency on the commercial data management system Xively. It can be assumed that it was used because Xively staff was actively involved in the development and it helped setting up the Air Quality network faster as not much effort had to be put into building the backbone data management system. The commercial system provides the database, a dashboard to view data and data feeds which can be used to extract the raw data. In the Google Group, concerns were already voiced whether viewable Air Quality Egg data will be restricted to 30 days which is already the case for Xively free accounts [42]. If issues continue from using Xively we might see a move to a complete open source system in the future.

3.9 Envboard

The TECO Envboard [3] is a research sensor platform in development from KIT that was introduced in 2012. This mobile device carries a wide variety of environmental and other sensors: it is equipped with sensors for temperature, humidity, atmospheric pressure, VOC, acceleration, sound, light, dust, UV, O_3, CO, NO_x and more optional sensors to choose from (see Table 9), for which there are footprints in place, but which are not populated in the standard revision.

The Envboard can log sensor readings to its internal memory or transmit them to a paired Bluetooth device. There are two modes of measurement: either sensor readings can be triggered individually (i.e. sensors can be polled) or the device can be set to measure continuously at a configurable interval. This can be done for a freely configurable subset of the sensors. Both the sensor readings

System:	**TECO Envboard**
Sensors:	MICS-2614 (O_3), MICS-4514 (CO, NO_x), GP2Y1010 (dust), SHT21 (temperature, humidity), iAQ-Engine (VOC, indirect: CO_2), WM-61A (custom, dBA noise level), TEPT5700b (ambient light), AlGaN-TO18 (UV light), MPL115A (atmospheric pressure), ADXL345 (3D accelerometer) Optional sensors (footprints ready): HMC5883 (3D gyroscope), ITG-3200 (3D magnetometer), MVS0608.02 ball switch (motion/microvibration), TGS4161 (CO_2), NTC thermistor (temperature), GPS module (global postion)
Additional features:	micro fans for air flow, solar panel
Dimensions:	ca. 7 x 14 x 3 cm
Price:	n/a
Availability:	n/a
Interface:	Bluetooth or Serial (Firmata-based Protocol)
Usage:	Stationary or mobile, triggered and/or continuous measurements, transmitted to host and/or stored locally on microSD card

Table 9. TECO Envboard overview

as well as the configuration of the device can be done over microUSB or the Bluetooth interface, which uses a protocol that is built as an extension of the open-source *Firmata* protocol. Using the microUSB port, the device can also be powered or its battery can be charged.

While there is no publicly available documentation on details of the Envboard's gas sensing capabilities (such as the accuracy or measurement frequency), in Air Quality research, the Envboard has been used for meaningful particulate matter measurements using a cheap, commercial off-the-shelf dust sensors [4]. To this aim, the authors first surveyed a variety of small dust sensors and selected the Sharp GP2Y1010 as most appropriate for their experiments [5]. Subsequently, the Envboard was used to devise calibration procedures that enable meaningful readings from the cheap GP2Y1010 sensor. While these procedures proved to be suitable for reaching a high accuracy, frequent de-calibration of the sensor's baseline remained a problem, which makes additional procedures necessary when using the GP2Y1010 for particulate matter measurements [7].

3.10 Smart Citizen

Smart Citizen is an open source air quality sensing project developed by members of Fab Lab Barcelona, Spain since 2009 [32]. Like the Air Quality Egg, the production run was funded by a Kickstarter campaign [79] which lasted for 30 days from May to June 2013. Much development work has been done and 150 beta sensor boards were already distributed to users around Barcelona to test hardware production, real world deployment and the web API. Using Kickstarter, the developers hope to distribute another 500 or more sensor boards to users around the world as well as finalize the design of a 3D-printed enclosure and the iOS app. If funds permit, an Android app and additional sensor shields for urban agriculture, electromagnetic fields, energy consumption, indoor air composition and biometrics will be in the works. All software, hardware and enclosure designs will be made available through Github repositories [80].

The Smart Citizen system consists of sensor boards (called *Smart Citizen Kit*), a data visualization website with API and a mobile phone app for iPhones. Sensor data management is handled by the commercial system Xively while user data is saved in the Smart Citizen database. Map data is provided by OpenStreetMap and the custom live map [81] currently uses a paid service called MapBox. Apart from API calls to Xively and MapBox, all the software is open source [80]. The iPhone app is under development and features include map visualizations similar to the website and live readings by connecting to the user's sensor board.

The hardware platform SensorBox is made up of two circuit boards: a data-processing (arduino-compatible) board and an interchangeable sensor shield. Both are connected and powered either by a battery or using a solar panel. This sensor board is supposed to measure different phenomena depending on the incorporated sensors with the air quality sensor board being the only one designed as of yet. It contains NO_2, CO, temperature/humidity, light and sound

System:	**Smart Citizen**
Sensors:	MiCS-2710 (NO_2), MICS-5525 (CO), DHT22 (temperature, humidity), sound, light
Additional features:	Solar panel
Dimensions:	5,6 x 6,7 cm (board only) [78]
Price:	$155 (assembled)
Availability:	Kickstarter campaign rewards
Interface:	Wi-Fi, miniUSB, web dashboard, iPhone app
Usage:	Stationary, continuous measurements

Table 10. Smart Citizen overview

sensors. Data is streamed via Wi-Fi using a WIFILY RN-131G card to the Xively servers, so the board can be placed anywhere as long as it has Wi-Fi access.

Boards were available by backing the Kickstarter campaign and no information was published whether they will be buyable in online shops at a later date. Several versions were offered as Kickstarter rewards: an unassembled kit with enclosure ($105), assembled board with enclosure ($155), assembled board with finalized enclosure and solar panel ($225).

Fab Lab Barcelona has created a participatory sensing system designed to support not only air quality sensing but also data from various other sources. As far as the air quality sensor board is concerned, the same problems exist like they did in the Air Quality Egg. For example the same NO, CO, temperature and humidity sensors are used, known for their low sensor data quality. So far, NO and CO readings are displayed on the live website [81] as $k\Omega$ values and not as readable gas concentrations. A different problem it has in common with the Air Quality Egg is the usage of commercial APIs although everything else is open source.

3.11 Gasser

Gasser is a sensor platform developed in 2012 by LaboCitoyen from France, an organization that creates open source hardware and software solutions for participatory sensing. They maintain a wiki [60] of their projects, with Gasser being one of them. While it appears that there are no plans to distribute these sensor platforms to the public as of yet, promising prototypes have already been built. The second prototype platform is stand-alone, mobile and contains an Alphasense NO2-B4 NO_2 sensor, GPRS and battery. A parts breakdown is given on the wiki page, listing a Raspberry Pi, Huawei E220 GPRS dongle, Alphasense B4 sensors, battery and enclosure, amounting to a total of €255. Alphasense B4 sensors were chosen for their precision for approximately €110 per sensor. The chosen 8Ah battery back lasts for about 5-6 hours and can be charged via micro USB.

Instead of streaming sensor data using GRPS, saving it to a SD card or transmitting it over Ethernet or 3G would also be possible due to the Raspberry

Pi interface. Other sensors can be added as the Alphasense B4 line [43] includes sensors for CO, NO, NO_2, SO_2 and H_2S.

Plans exists for releasing tutorials and Raspberry Pi images for users to build their own Gasser sensor platforms. LaboCitoyen is also working on an open source sensor data management system called thingstream [83].

3.12 Sensordrone

Sensordrone [73] is a portable sensor platform developed in 2012 by Sensorcon, a sensor and electronics company from Buffalo, New York. The first production batch of around 900 Sensordrones was funded by the Kickstarter campaign [74] in June/July 2012 which managed to raise more than four times the original goal of $25,000. Now these matchbox sized sensor platforms can be bought directly from the Sensorcon website for $199 apiece and the developers are working on extensions like the CO_2 module which is currently up for preorder. Sensorcon also offers portable low-cost CO detectors and H_2S detectors for personal and industrial use.

Sensordone is a pure sensor platform and is not tied to a certain application or air quality sensing system. It provides a Bluetooth interface to which other devices (e.g. mobile phones) can connect to. The underlying protocol for communicating with the Sensordrone has not been revealed yet but the developers are considering releasing it. The same applies to future firmware updates which might enable communication via USB for users without mobile phones. Documentation for the Android API [75] and libraries can be found on the Sensordrone Developers page for users to write their own Android apps. So far, Sensorcon has released five example Android apps which display sensor readings from the Sensordrone. Sources of all apps listed here can be found on Github [76]:

- *Sensordrone Control*, the first application displays live readings from all sensors except for the reducing gas sensor and the oxidizing gas sensor, because they are high in energy consumption. Single sensor reading streams can be turned on or off and a simple graphing function plots the last 30 measurements.

System:	**Gasser**
Sensors:	Alphasense NO2-B4 (NO_2)
Additional features:	-
Dimensions:	approx. 10 x 15 x 5 cm [60]
Price:	€255
Availability:	not available
Interface:	GPRS
Usage:	Mobile

Table 11. Gasser overview

- *Virtual CO Inspector* mimics the interface of Sensorcon's *CO* detector and shows *CO* readings in ppm using the precision gas sensor.
- *Crisper Humidity Guide* can be used to monitor humidity in the refrigerator. Users can then adjust the humidity until it's at the correct level for the products stored.
- *Color Intensity Meter* shows RGB readings from the color photodiodes and calculates illuminance in Lux from the RGB values.
- *Infrared Thermometer* is a simple application for measuring surface temperature and comparing it to a stated reference value.

The sensor platform hardware contains a Bluetooth module, four sensor clusters, selectable colored LEDs and connector pins to attach your own sensors or hardware. It can not be turned off but will not power its sensors unless a connected Bluetooth device requests readings. Therefore the Bluetooth standby will still drain the battery while it is not in use but should last a few days. The Micro USB port is currently only for charging the battery which needs one to two hours until fully charged.

A list of specific sensors used in the Sensordrone is not available but a specification document can be found in the Sensordrone Developers forum [77], listing several properties for each sensor, mostly concerning accuracy and power consumption. Table 12 shows a summary of the sensors specification. On the device, these sensors are arranged in three clusters in the front of the device and one sensor in the back. The sensors are distributed as follows:

- Air sensor port: precision gas sensor, reducing gas sensor, oxidizing gas sensor, temperature sensor, humidity sensor, pressure sensor
- Infrared temperature sensor
- Light sensors: red intensity sensor, green intensity sensor, blue intensity sensor, clear intensity sensor
- Proximity capacitance sensor (backside)

Sensorcon has equipped the Sensordrone with a total of twelve sensors to allow a wide range of applications for users to create. Staff members also frequent the Developers forum to answer questions and provide updates and ideas for existing apps and the Sensordrone firmware. The Sensordrone itself however is less suited for air quality sensing systems which go beyond detecting "good" or "bad" air quality. According to the specifications, the reducing gas sensor and the oxidizing gas sensors are sensitive to many gases and therefore not capable of determining concentrations of certain gases. They also consume too much power to run for longer periods of time and significantly influence other sensor readings due to heat. Although the small size and ease of use would go well with any mobile participatory sensing system, whether Sensorcon can release a Sensordrone with sensors that can provide more accurate data for air quality sensing remains to be seen.

Precision gas sensor: pre-calibrated for CO, sensitive to a variety of gases		
Sensor type: electrochemical	Power consumption: low	Resolution: 1ppm
Range: 0-2000ppm	Accuracy: $+/-10\%$	Response time: 10-20s
Reducing gas sensor: reacts to e.g. H_2, CO, VOCs, needs 2-3 min warmup		
Sensor type: metal oxide	Power consumption: high	Resolution: nA
Range: 0-1000ppm	Accuracy: nA	Response time: 30-60s
Oxidizing gas sensor: reacts to e.g. O_3, NO_2, Cl_2, needs 2-3 min warmup		
Sensor type: metal oxide	Power consumption: high	Resolution: nA
Range: 0-5ppm	Accuracy: nA	Response time: 30-60s
Temperature		
Sensor type: silicon bandgap	Power consumption: low	Resolution: nA
Range: $-20°C$ to $+60°C$	Accuracy: $+/-0.5°C$	Response time: 20-60s
Humidity: in relative humidity (RH)		
Sensor type: capacitive	Power consumption: low	Resolution: nA
Range: $0-100\%$ RH	Accuracy: $+/-2\%$ RH	Response time: 10-180s
Pressure		
Sensor type: MEMS	Power consumption: low	Resolution: $1.5Pa$
Range: $30-110kPa$	Accuracy: $+/-0.1kPa$	Response time: $1s$
Infrared temperature: non-contact, surface temperature		
Sensor type: thermopile	Power consumption: med.	Resolution: nA
Range: $-20°C$ to $+60°C$	Accuracy: $+/-1°C$ to $3°C$	Response time: $1-5s$
Proximity capacitance		
Sensor type: electrodes	Power consumption: med.	Resolution: $0.5fF$
Range: $0-4pF$	Accuracy: nA	Response time: $< 1s$
Color: red, green, blue and clear illumination		
Sensor type: photodiode	Power consumption: high	Resolution: nA
Range: nA	Accuracy: nA	Response time: $< 1s$

Table 12. Sensordrone sensors specification

4 Discussion

Air quality sensing carries many issues that extend beyond building a reliable sensor platform and that all need to be addressed in a fully working system. Problems involved can be roughly categorized in application issues and sensor platform issues. We have covered twelve sensor platforms, their respective architectures and their provided applications. All systems we have covered deal with only a part of these issues and even ambitious projects like CitiSense (see Section 3.2) have planned to solve many problems but did not include all solutions in their prototype systems because of the work involved. However, many interesting concepts have been proposed that often can be applied to participatory sensing or air quality sensing in general. We discuss the different issues in view of their problems and solutions in the following paragraphs.

System:	**Sensordrone**
Sensors:	Precision gas, oxidizing gas, reducing gas, temperature (ambient and infrared), humidity, pressure, proximity capacitance, color (RGB), light
Additional features:	Connectors to add own sensors
Dimensions:	6.8 x 2.8 x 1.2 cm
Price:	$200
Availability:	Sensorcon website
Interface:	Bluetooth, Android API + apps
Usage:	Mobile

Table 13. Sensordrone overview

4.1 Sensor platform

Sensor platforms today still have problems concerning price, data quality, mobility (only in mobile applications), power management and usability. It's not reasonable to skim on any of these concerns in a system that targets real users except for data quality. Even if problems are singled out, not all of them can be solved satisfactorily.

All sensor platforms we presented for which information on the price is available cost less than $1000, the most expensive being the CitiSense SEEnsor node and the AirBase CanarIT. While this price is still too high for participatory sensing, it shows that the developers of all these systems tried to keep the price low. More desirable prices (around $200) are achieved by AirQualityEgg, SmartCitizen and Sensordrone.

Projects like AirQualityEgg (see Section 3.8) purposely sacrifice data quality by using small low-cost sensors without calibration, thus allowing more people to buy the sensor platform. The developers consider involving more people into participating more important than raising accurate data and leave this problem to future work. Especially gaseous chemicals are hard to measure with low-cost sensors and even more accurate sensors in a higher price range like the Alphasense $CO - BF$ sensor (>€100) are sensitive to temperature changes [13]. To compensate such unwanted influences, each sensor needs to be calibrated individually which increases price and maintenance effort. A solution to mitigate sensor noise is proposed by EveryAware (see Section 3.3) by using a sensor array with multiple low-cost sensors for one gas and automatic calibration. For particulate matter measurement, Dylos (see Section 3.5) is already providing an affordable and accurate system for consumers. The success of the Dylos Air Quality Monitor encourages sensor platforms that specialize on measuring specific substances instead of platforms that measure a wide range of substances but do so poorly.

Another big concern is mobility for mobile sensor platforms which means first and foremost keeping size and weight low. Sensor platforms like those used in CitiSense or AirCasting do not exceed the size of a big handheld system but still need to become smaller, preferrably around the size of the Sensordrone (see

Section 3.12) so they can be carried around conveniently. Power management, short stabilization time and ensuring enough airflow are additional issues that arise when building mobile platforms. An included battery should at least last through a day so the user only needs to charge it overnight like a smart phone. To save power, sensor platform could only sample data when necessary or allow local sharing of sensor readings via Bluetooth like the MAQS system (see Section 3.1), given enough participants. Because of these problems, there is no consumer ready mobile platform yet. The Sensordrone has an attractive size but can not power its gas sensors for a full day of usage and is therefore not suited for air quality measuring.

The systems that are currently deployed ready for consumers use stationary sensor platforms either for personal sensing (e.g. Dylos) or participatory sensing (e.g. AirBase, AirQualityEgg, SmartCitizen). Even though they all have unsolved issues, air quality sensing has found its way into consumer products and more, sometimes novel, projects are brought into being today like CubeSensors or iSpex.

We also identified three different measuring techniques (see Section 2.1) to directly or indirectly infer air quality. The indirect methods aim at removing the need for expensive sensor hardware and the problems that come along with it but there are very few projects exploring indirect measurement. Therefore, not much can be said about the data quality that can be achieved with those methods. iSpex is a current running project using sky observation for air quality measurement so indirect methods - although rare - are still pursued.

4.2 Architecture/Application

Apart from the sensor platform, a participatory sensing system needs an architecture and applications to process, aggregate and visualize sensor data.

Participatory air quality sensing systems rely on the input of a large number of active users to produce meaningful pollution maps and to detect patterns in how air quality develops. With so many different projects started by governments, researchers, hobbyists and companies, we assume a system that can sufficiently cover larger pollution maps with data will incorporate sensor data streams from many different sensor platforms. Two projects (CitiSense and EveryAware) suggest to include official measurement stations for reference data and calibration as mobile sensor platforms pass by. If all data is gathered by one data management system, a distributed, open source solution would be preferrable. This way, different interest groups can set up their own storage without depending on a single company and users can choose the storage they trust the most. The recent troubles of AirQualityEgg with the commercial Xively storage strengthen this concern. A data management system like AirCasting (see Section 3.7) which allows arbitrary data channels (e.g. heart rate) could be the answer to a heterogenous mixture of sensor platforms used. This would also work well with many specialized, accurate sensor platforms such as the Dylos Air Quality Monitor. Different users could provide different air quality parameters which form one coherent pollution map.

Another reason in favor of a data management system that can support data from many sensor platforms is that air quality sensing includes different use cases such as indoor/outdoor or mobile/stationary measurements. Each has its own characteristics and system requirements so a single sensor platform probably can not be used for all use cases. MAQS for example is a system for mobile indoor air quality sensing which uses Wi-Fi fingerprints for identifying rooms. This can be useful not only for personal sensing at home but for indoor pollution maps in public buildings like schools or hospitals. CitiSense wants to automatically classify sensor readings as indoors or outdoors so the sensor platform can be used in both settings.

None of the management systems actively deal with privacy and security concerns. Some systems have simple access control features to fully hide or share sensor readings publicly (e.g. MAQS, AirQualityEgg). When a system gets more users and more data to maintain this will become a bigger problem than it is right now. A main concern is the possibility of tracking a certain user's movements but as we've mentioned before, no mobile platform is quite ready yet. EveryAware plans to maintain user privacy by adding noise to the data and by implementing a more fine-grained access control system. An interesting approach is to have the smart phone app process data and provide statistics without communicating with a back-end server. This is sufficient for personal sensing in case the user does not want to disclose his data.

Personal sensing is very similar to participatory sensing but has less problems, such as privacy issues. It should be possible to use sensor platforms designed for participatory sensing in personal sensing applications as the only changes that need to be made are in software. These changes include streaming data to the user's hardware only and keeping tabs on the user's exposure to pollution instead of simply saving sensor readings.

5 Conclusion

A lot of work has already been done on (participatory) air quality sensing, especially in the last five years. Different stakeholders such as researchers, hobbyists and companies have tried their approaches on sensor platforms and data management backends. Air quality sensing has also piqued the interest of environment-conscious users which is why air quality sensing systems will continue to be developed for consumers. A few working stationary systems available for purchase already exist. We covered twelve different air quality sensing projects and many interesting concepts have been proposed, mostly by researchers to solve certain problems. Low-cost sensors and rapid prototyping hardware are readily available today for everyone to build their own sensor platforms but problems like data quality and mobility still remain. For participatory sensing, a necessary open data management backend needs to be in place to relieve each new projects from developing their own so they can concentrate on improving their sensor platform. It is hoped that such a backend can support the work and

data of different projects from different stakeholders in a heterogenous sensor environment.

References

1. Aberer, K., Sathe, S., Chakraborty, D., Martinoli, A., Barrenetxea, G., Faltings, B. and Thiele, L.: OpenSense: Open community driven sensing of environment, Proceedings of the ACM SIGSPATIAL International Workshop on GeoStreaming, 39–42 (2010)
2. Bales, E., Nikzad, N., Quick, N., Ziftci, C., Patrick, K. and Griswold, W., Citisense: Mobile air quality sensing for individuals and communities Design and deployment of the Citisense mobile air-quality system, Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2012 6th International Conference on, 155–158 (2012)
3. Budde, M., Berning, M., Busse, M., Miyaki, T. and Beigl, M., The TECO Envboard: A mobile sensor platform for accurate urban sensing – and more, 9th International Conference on Networked Sensing Systems (INSS 2012), IEEE (2012)
4. Budde, M., Berning, M., Busse, M., Miyaki, T. and Beigl, M., Handheld Particulate Matter Measurements with COTS Sensors 10th International Conference on Pervasive Computing (Pervasive 2012)
5. Budde, M., Busse, M., and Beigl, M., Investigating the Use of Commodity Dust Sensors for the Embedded Measurement of Particulate Matter, 9th International Conference on Networked Sensing Systems (INSS 2012), IEEE (2012)
6. Budde, M., Barbera, P., El Masri, R., Riedel, T. and Beigl, M., Retrofitting Smartphones to be Used as Particulate Matter Dosimeters, 17th International Symposium on Wearable Computers (ISWC'13), 139–140
7. Budde, M., El Masri, R., Riedel, T. and Beigl, M., Enabling Low-Cost Particulate Matter Measurement for Participatory Sensing Scenarios, 12th International Conference on Mobile and Ubiquitous Multimedia (MUM'13), *to appear* (2013)
8. Burke, J.A., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S. and Srivastava, M.B.: Participatory sensing, Center for Embedded Network Sensing, University of California (2006)
9. De Cristofaro, E. and Soriente, C.: Participatory privacy: Enabling privacy in participatory sensing, IEEE Network 27, 32–36 (2013)
10. Demirbas, M., Rudra, C., Rudra, A. and Bayir, M.A.: imap: Indirect measurement of air pollution with cellphones, Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on, 1–6 (2009)
11. Dua, A., Bulusu, N. and Feng, W. and Hu, W.: Towards trustworthy participatory sensing, Proceedings of the 4th USENIX conference on Hot topics in security (2009)
12. Elen, B., Theunis, J., Ingarra, S., Molino, A., Van den Bossche, J., Reggente, M. and Loreto, V., The EveryAware SensorBox: a tool for community-based air quality monitoring (2012)
13. Fondazione ISI, Turin, Italy.: EveryAware: Enhance Environmental Awareness through Social Information Technologies, Report on: sensor selection, calibration and testing; EveryAware platform; smartphone applications (2012)
14. Hasenfratz, D., Saukh, O., Sturzenegger, S. and Thiele, L.: Participatory air pollution monitoring using smartphones, Proc. 1st Intl Workshop on Mobile Sensing: From Smartphones and Wearables to Big Data (2012)
15. Hayes M., Calleja M. and Jones R.: CamMobSens: Cambridge Mobile Urban Sensing `http://www.escience.cam.ac.uk/mobiledata/` (2010)

16. Honicky, R., Brewer, E.A., Paulos, E. and White, R.: N-smarts: networked suite of mobile atmospheric real-time sensors, Proceedings of the second ACM SIGCOMM workshop on Networked systems for developing regions, 25–40 (2008)
17. Hooker, B., Gaver, W., Steed, A. and Bowers, J.: The pollution e-sign, Workshop on Ubiquitous Sustainability. Ubicomp (2007)
18. Jiang, Y., Li, K., Tian, L., Piedrahita, R., Yun, X., Mansata, O., Lv, Q., Dick, R.P., Hannigan, M. and Shang, L., MAQS: a personalized mobile sensing system for indoor air quality monitoring, Proceedings of the 13th international conference on Ubiquitous computing, 271–280, 2011
19. Jones R., and Calleja M.: On the Road with Mobile Sensors, `http://www.thenakedscientists.com/HTML/content/interviews/interview/1107/`
20. Kuznetsov, S. and Paulos, E.: Participatory sensing in public spaces: activating urban surfaces with sensor probes, Proceedings of the 8th ACM Conference on Designing Interactive Systems, 21–30 (2010)
21. Kuznetsov, S., Davis, G., Cheung, J. and Paulos, E.: Ceci n'est pas une pipe bombe: authoring urban landscapes with air quality sensors, Proceedings of the 2011 annual conference on Human factors in computing systems, 2375–2384 (2011)
22. Lane, N.D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T. and Campbell, A.T., A survey of mobile phone sensing, Communications Magazine 48, IEEE, 140–150 (2010)
23. Mendez, D., Perez, A.J. and Labrador, M.A. and Marron, J.J.: P-sense: a participatory sensing system for air pollution monitoring and control, Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on, 344–347 (2011)
24. Mun, M., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., Hansen, M., Howard, E., West, R. and Boda, P.: PEIR, the personal environmental impact report, as a platform for participatory sensing systems research, Proceedings of the 7th international conference on Mobile systems, applications, and services, 55–68 (2009)
25. Nikzad, N., Verma, N., Ziftci, C., Bales, E., Quick, N., Zappi, P., Patrick, K., Dasgupta, S., Krueger, I., Rosing, T.S. and others, CitiSense: improving geospatial environmental assessment of air quality using a wireless personal exposure monitoring system, Proceedings of the conference on Wireless Health (2012)
26. Nikzad, N., Ziftci, C., Zappi, P., Quick, N., Aghera, P., Verma, N., Demchak, B., Patrick, K., Shacham, H., Rosing, T.S. and others, CitiSense: Adaptive Services for Community-driven Behavioral and Environmental Monitoring to Induce Change (2011), Department of Computer Science and Engineering, University of California, San Diego
27. Poduri, S., Nimkar, A. and Sukhatme, G.S.: Visibility monitoring using mobile phones, Technical report, Department of Computer Science, University of Southern California (2010)
28. Steed, A., Spinello, S., Croxford, B. and Greenhalgh, C.: e-Science in the streets: urban pollution monitoring, UK e-science all hands meeting (2003)
29. Stevens, M. and DHondt, E.: Crowdsourcing of pollution data using smartphones, Workshop on Ubiquitous Crowdsourcing (2010)
30. Whitney, M. and Richter Lipford, H.: Participatory sensing for community building, CHI '11 Extended Abstracts on Human Factors in Computing Systems, 1321-1326 (2011)
31. Zappi, P., Bales, E., Park, J.H., Griswold, W. and Rosing, T., The CitiSense air quality monitoring mobile sensor node (2012), Proceedings of the 11th ACM/IEEE Conference on Information Processing in Sensor Networks, Beijing, China

32. Acrobotic Industries Smart Citizen Overview, http://acrobotic.com/smart-citizen
33. AirBase, http://www.myairbase.com
34. AirBase Sensors Map, http://sensors.myairbase.com/
35. AirBase Sensor Data Download, http://sensors.myairbase.com/Sensors/xls/1
36. AirCasting, http://aircasting.org
37. AirNow, U.S. EPA Office of Air Quality Planning and Standards, http://airnow.gov
38. Air Quality Egg, http://airqualityegg.com/
39. Air Quality Egg Blog, http://aqe.wickeddevice.com
40. Air Quality Egg Kickstarter Campaign, http://www.kickstarter.com/projects/edborden/air-quality-egg
41. Air Quality Egg Wiki, http://airqualityegg.wikispaces.com
42. Air Quality Egg Google Group, https://groups.google.com/forum/?fromgroups#!forum/airqualityegg
43. Alphasense B4 sensors, http://www.alphasense.com/industrial-sensors/alphasense_sensors/B4_sensors.html
44. AQE: You can help build an open air quality sensor network, http://blog.xively.com/2011/12/07/you-can-help-build-an-open-air-quality-sensor-network/
45. AQE Wicked Device, http://wickeddevice.com/index.php?main_page=index&cPath=28
46. Air Quality in Europe, CITEAIR, http://www.airqualitynow.eu
47. CitizenSensor, http://thesis.jmsaavedra.com/
48. Citizen Mobile Environmental Sensing, http://www.ecomobilecitizen.com/
49. Council Directive 1999/30/EC of 22 April 1999 relating to limit values for sulphur dioxide, nitrogen dioxide and oxides of nitrogen, particulate matter and lead in ambient air
50. CubeSensors, http://www.cubesensors.com
51. CubeSensors Blog, http://blog.cubesensors.com
52. Dylos Corporation, http://www.dylosproducts.com
53. Dylos Reviews,
http://www.air-purifier-power.com/dylosdc1100proreview.html,
http://www.revieweur.com/dylos-dc1100-laser-air-quality-monitor-review.html,
http://www.americanallergysupply.com/dylos/
54. EveryAware Air Quality Case Study,
http://www.everyaware.eu/activities/case-studies/air-quality/
55. EveryAware AirProbe Sensor Map,
http://cs.everyaware.eu/event/airprobe/map2
56. FacultyRow: Small, Portable Sensors Allow Users to Monitor Exposure to Pollution on Their Smart Phones, http://facultyrow.com/profiles/blogs/ucsd-innovation-measuring-air-pollution-with-iphones
57. Flickr CitiSense Photo Set,
http://flickr.com/photos/jsoe/sets/72157632248508332/with/8273330584/
58. HabitatMap, http://habitatmap.org
59. iSPEX, http://ispex.nl/en/ (2013)
60. LaboCitoyen Wiki, http://wiki.labocitoyen.fr/index.php?n=Hardware.Gasser
61. La montre verde (The green watch), CityPulse, Fing http://fing.org/-Green-Eyes-Montre-verte-CityPulse-?lang=fr (2009)

62. Landesanstalt für Umwelt, Messungen und Naturschutz Baden-Württemberg, http://mnz.lubw.baden-wuerttemberg.de/messwerte/aktuell/
63. Matter Aerosol DiSCmini, http://www.matter-aerosol.ch/index.php/features/jquery-superfish-menu
64. MAQS Wiki, http://maqs.pbworks.com/
65. MobileDust, http://www.teco.edu/research/mobiledust/
66. netatmo, http://www.netatmo.com
67. OpenSense ETH Zürich, http://www.opensense.ethz.ch/trac/
68. OpenSense Live Data Browser, http://data.opensense.ethz.ch/
69. PM-Air, Preemptive Media (Beatriz da Costa, Jamie Schulte and Brooke Singer), http://www.pm-air.net (2006)
70. PM-Air readings, http://www.pm-air.net/dataVis/ (accessed 19/05/2013)
71. Project Budburst, http://budburst.org/
72. Sensaris, http://www.sensaris.com
73. Sensordrone, http://www.sensorcon.com/sensordrone-1/
74. Sensordrone Kickstarter Campaign, http://www.kickstarter.com/projects/453951341/sensordrone-the-6th-sense-of-your-smartphoneand-be
75. Sensordrone Android API, http://developer.sensordrone.com/android/api/
76. Sensordrone Android Apps Github, https://github.com/Sensorcon
77. Sensordrone Developers Forum, http://developer.sensordrone.com/forum/
78. Smart Citizen, http://smartcitizen.me/
79. Smart Citizen Kickstarter Campaign, http://www.kickstarter.com/projects/acrobotic/the-smart-citizen-kit-crowdsourced-environmental-m
80. Smart Citizen Github Repository, https://github.com/fablabbcn
81. Smart Citizen Live Beta, http://test.smartcitizen.me/
82. Statistisches Landesamt Baden-Württemberg, http://www.statistik.baden-wuerttemberg.de/Pressemitt/2013160.asp
83. thingstream, http://www.thingstream.net
84. TSI Dust Monitors, http://www.tsi.com/Dust-Monitors/
85. United States Environmental Protection Agency http://www.epa.gov/airquality/