

# **Context as a Service**

Dissertation

in partial fulfillment of the requirements for the degree

**Doktor der Naturwissenschaften (Dr. rer. nat.)**

for submission to the

Faculty of Electrical Engineering and Computer Science

University of Kassel, Germany

Kassel 2012

Dipl.-Inf. Michael Wagner

**Advisors:**

Prof. Dr. Kurt Geihs, Universität Kassel

Prof. Dr. Christian Becker, Universität Mannheim

**Additional Doctoral Committee Members:**

Prof. Dr. Klaus David, Universität Kassel

Prof. Dr. Arno Wacker, Universität Kassel

**Date of Defense:** 4 March 2013



*To Anke and Jakob*



# Contents

---

<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Listings</b>	<b>ix</b>
<b>List of Algorithms</b>	<b>xi</b>
<b>List of Abbreviations</b>	<b>xiii</b>
<b>Abstract</b>	<b>xv</b>
<b>Zusammenfassung</b>	<b>xvii</b>
<b>Acknowledgements</b>	<b>xix</b>
<b>I Foundations</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Motivation . . . . .	3
1.2 Problem Statement . . . . .	6
1.3 Solution Approach . . . . .	8
1.4 Contribution . . . . .	10
1.5 Structure of the Thesis . . . . .	10
<b>2 Ubiquitous Computing Systems</b>	<b>13</b>
2.1 Ubiquitous Computing . . . . .	13
2.2 Context . . . . .	16
2.2.1 Context Awareness . . . . .	16
2.2.2 Characteristics of Context Data . . . . .	17
2.2.3 Quality of Context . . . . .	18
2.2.4 Cost of Context . . . . .	23
2.3 Self-Adaptation . . . . .	24
2.3.1 Adaptation Mechanisms . . . . .	24
2.3.2 Adaptation Policies . . . . .	25
<b>3 Logic and Ontologies</b>	<b>27</b>
3.1 Proof Methods for Propositional Logic . . . . .	28
3.1.1 Analytic Tableaux for Propositional Logic . . . . .	28
3.1.2 Resolution for Propositional Logic . . . . .	32

3.2	Proof Methods for First-Order Logic . . . . .	33
3.2.1	Resolution for First-Order Logic . . . . .	33
3.2.2	Analytic Tableaux for First-Order Logic . . . . .	36
3.3	Ontologies . . . . .	37
3.3.1	General Discussions on Ontologies . . . . .	38
3.3.2	OWL . . . . .	40
<b>4</b>	<b>Related Work</b>	<b>43</b>
4.1	Existing Context-Aware Systems . . . . .	44
4.1.1	Adaptive middleware for context-aware applications in smart-homes	45
4.1.2	ASC-CoOL & CoCo . . . . .	45
4.1.3	AWARENESS . . . . .	46
4.1.4	CARE . . . . .	47
4.1.5	CoBrA . . . . .	48
4.1.6	CONTEXT . . . . .	48
4.1.7	Context Toolkit . . . . .	49
4.1.8	COSMOS . . . . .	50
4.1.9	EEMSS . . . . .	51
4.1.10	Gaia . . . . .	52
4.1.11	Hydrogen . . . . .	52
4.1.12	Information exchange and fusion in dynamic and heterogeneous distributed environments . . . . .	53
4.1.13	Managing Context Information in Mobile Devices . . . . .	54
4.1.14	MobiLife . . . . .	55
4.1.15	MUSIC & Paspallis . . . . .	55
4.1.16	Nexus . . . . .	56
4.1.17	Quality-Aware Context Management Middleware (QCMM) . . . . .	56
4.1.18	Sentient Object Model . . . . .	57
4.1.19	SOCAM . . . . .	58
4.1.20	Supporting pervasive computing applications with active context fusion and semantic context delivery . . . . .	58
4.2	Summary . . . . .	59
<b>II</b>	<b>Solution Approach</b>	<b>63</b>
<b>5</b>	<b>Overview</b>	<b>65</b>
<b>6</b>	<b>Context Model</b>	<b>69</b>
6.1	Layers of the Context Model . . . . .	69
6.2	Context Information . . . . .	71
6.3	Metadata . . . . .	74
6.4	Operations . . . . .	76
6.4.1	Inter-Representation Operations . . . . .	77
6.4.2	Metadata Operations . . . . .	79
6.5	Hierarchical Composition of the Ontology . . . . .	79
6.6	Discussion . . . . .	80

<b>7</b>	<b>Context Offer and Query Language</b>	<b>83</b>
7.1	Context Offer and Request . . . . .	84
7.2	Constraints . . . . .	85
7.3	Selection Function . . . . .	86
7.4	Example . . . . .	88
7.5	Discussion . . . . .	90
<b>8</b>	<b>Context Offer and Query Matching</b>	<b>91</b>
8.1	Initial Matching . . . . .	92
8.2	Mediation Check . . . . .	93
8.3	Metadata Constraint Matching . . . . .	101
8.4	Example . . . . .	106
8.5	Discussion . . . . .	109
<b>9</b>	<b>Context Service Selection</b>	<b>113</b>
9.1	Motivating Example . . . . .	114
9.2	The Selection Approach . . . . .	116
9.2.1	Syntactic Elements . . . . .	116
9.2.2	Semantics . . . . .	118
9.3	Calculation of the Domain of a Quality or Cost Dimension . . . . .	122
9.3.1	Example . . . . .	125
9.4	Selection Algorithm . . . . .	126
9.5	Example . . . . .	130
9.6	Discussion . . . . .	135
<b>10</b>	<b>Architecture</b>	<b>139</b>
<b>III</b>	<b>Evaluation</b>	<b>145</b>
<b>11</b>	<b>Demonstrators</b>	<b>147</b>
11.1	Study of Context Information, QoC, and CoC in Related Work . . . . .	148
11.2	General Description of the Demonstrators . . . . .	152
11.3	Demonstrator A: Heterogeneity . . . . .	158
11.4	Demonstrator B: Simple Selection . . . . .	163
11.5	Demonstrator C: Multiple Selection . . . . .	170
11.6	Demonstrator D: Discovery of Remote Offers . . . . .	172
11.7	Demonstrator E: Cost Minimization . . . . .	173
<b>12</b>	<b>Performance and Scalability</b>	<b>175</b>
12.1	Mediation Service . . . . .	175
12.2	Constraint Matching . . . . .	182
12.3	Selection Service . . . . .	185
<b>13</b>	<b>Conclusions</b>	<b>189</b>
13.1	Summary of Contributions . . . . .	189
13.2	Outlook and Future Work . . . . .	192

<b>IV Appendices</b>	<b>195</b>
<b>A Logging Output for Demonstrator B</b>	<b>197</b>
<b>B Erklärung</b>	<b>199</b>
<b>C Bibliographies</b>	<b>201</b>
C.1 Bibliography . . . . .	201
C.2 Publications as (Co-)Author . . . . .	215



# List of Figures

---

1.1	Overview of the Solution Approach. . . . .	9
2.1	Taxonomy of Computer Systems Research Problems in Pervasive Computing	14
2.2	Interdependence between QoS, QoS and QoD . . . . .	19
2.3	QoC Processing Model . . . . .	21
2.4	States and Actions . . . . .	25
3.1	Analytic Tableaux for Different Construction Rules. . . . .	29
3.2	Analytic Tableau for $\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$ . . . . .	31
3.3	Analytic Tableau for $(\forall x)(Px \rightarrow Qx) \rightarrow ((\forall x)Px \rightarrow (\forall x)Qx)$ . . . . .	37
3.4	Semantic Web Layer Cake . . . . .	40
3.5	RDF Statement and Example . . . . .	41
5.1	Detailed Overview of the Solution Approach including Stakeholders. . . . .	66
6.1	Layers of the Context Model . . . . .	70
6.2	General Concepts: Entity, Scope and Representation . . . . .	71
6.3	Several Examples for Multiple Representations . . . . .	72
6.4	Detailed Example for a Composite Representation . . . . .	73
6.5	Context Metadata . . . . .	75
6.6	Context Operations . . . . .	77
6.7	Hierarchical Composition of the Ontology . . . . .	80
7.1	Overview of the Context Offer and Query Language (COQL) . . . . .	84
7.2	COQL: Subscription Modes . . . . .	85
7.3	COQL: Operators . . . . .	86
7.4	COQL: Entity Constraints . . . . .	86
7.5	COQL: Scope Constraints . . . . .	87
7.6	COQL: Metadata Constraints . . . . .	87
7.7	COQL: Selection Function . . . . .	88
8.1	Overview on the Context Offer and Query Matching . . . . .	91
8.2	Mediator Chain Overview . . . . .	94
8.3	Example for Metadata Constraint Matching . . . . .	105
8.4	Example for the Context Offer and Query Matching . . . . .	106
8.5	Matching Example – Metadata Constraint Matching . . . . .	107
8.6	Mediator Chain Example . . . . .	108
8.7	Matching Example – Metadata Constraint Matching 2 . . . . .	109
9.1	Motivating Example for the Selection . . . . .	115
9.2	Example of a Selection Process – Offers and Queries . . . . .	132

10.1	Context Service Infrastructure and Execution Environment . . . . .	139
10.2	Context Service Middleware Architecture . . . . .	141
10.3	Screenshot of the Android Visualization Application . . . . .	142
10.4	Context Service Lifecycle . . . . .	144
11.1	Demonstrator A: Chains in Scenario 1. . . . .	161
11.2	Demonstrator A: Screenshots of the Visualization Application in the first Scenario. . . . .	162
11.3	Demonstrator A: Chains in the Second Scenario . . . . .	163
11.4	Demonstrator A: Screenshot of the Visualization Application for the Second Scenario. . . . .	164
11.5	Demonstrator B: Potential Mediator Chains in the First Scenario . . . . .	165
11.6	Demonstrator B: Screenshots of the Visualization Application in the First Scenario . . . . .	168
11.7	Demonstrator B: Potential Mediator Chains/Offers in the Second Scenario	169
11.8	Demonstrator D: Remote Discovery of Context Offers on the Second Device	173
11.9	Demonstrator E: Power Consumption in Different Use Cases . . . . .	174
12.1	Scalability Evaluation: Number of Representations Compared to Generated IROs . . . . .	176
12.2	Scalability Evaluation: Resulting Graph for Four Representations . . . . .	177
12.3	Scalability Evaluation: Number of Chains and Cycle Exceptions Depending on Number of IROs . . . . .	178
12.4	Scalability of the Establishment of Mediator Chains. . . . .	179
12.5	Memory Usage for the Establishment of Mediator Chains . . . . .	180
12.6	Mean Memory Usage per Normalized Mediator Chain . . . . .	181
12.7	Comparison of the Memory Usage on Different Devices during the Establishment of Mediator Chains . . . . .	182
12.8	Scalability of Matching Metadata Constraints. . . . .	183
12.9	Memory Usage while Matching Metadata Constraints . . . . .	184
12.10	Mean Runtime of the Selection . . . . .	187
12.11	Mean Runtime of the Selection Grouped by Number of Offers . . . . .	188
12.12	Mean Memory Usage of the Selection Process . . . . .	188

# List of Tables

---

2.1	Quality of Context Parameters . . . . .	23
3.1	Operation (A) for Construction of an Analytic Tableau . . . . .	30
3.2	Operation (B) for Construction of an Analytic Tableau . . . . .	30
4.1	Overview on Existing Context-aware Systems . . . . .	61
8.1	Definition of the Metadata Domain . . . . .	104
9.1	Example for Domain Calculation – Historical Values . . . . .	126
9.2	Example for Domain Calculation – Intermediate Results . . . . .	127
9.3	Selection Example 1 – Intermediate Results . . . . .	133
9.4	Selection Example 1 – Results . . . . .	134
9.5	Selection Example 2 – Results . . . . .	135
9.6	Selection Example 3 – Results . . . . .	136
11.1	Sensor Types Supported by the Android Platform . . . . .	149
11.2	Battery Consumption of the Accelerometer on a Google Nexus One . . . . .	149
11.3	Activity Detection Time in the Activity Recognition Approach by Wang et al. . . . .	151
11.4	Evaluation of Accuracy and Power Consumption in the Activity Recognition Approach by Yan et al. . . . .	152
11.5	Context Offers of the Implemented Context Services . . . . .	156
11.6	Context Queries of the Implemented Context Services . . . . .	157
11.7	Demonstrator A: Context Queries . . . . .	158
11.8	Demonstrator A: Potential Context Offers for Scenario 1 and 2. . . . .	159
11.9	Demonstrator A: Inter-Representation Operations . . . . .	160
11.10	Demonstrator A: Metadata Operation . . . . .	160
11.11	Demonstrator C: Context Queries . . . . .	170
11.12	Demonstrator C: Mediator Chains . . . . .	171
12.1	Scalability Evaluation: Generated IROs for Four Representations . . . . .	176
12.2	Runtime Statistics for Mediation . . . . .	179
12.3	Memory Usage for Mediation . . . . .	180
12.4	Memory Usage for Mediation (Desktop Computer) . . . . .	181
12.5	Memory Usage for Constraint Matching of 500 Constraints . . . . .	184
12.6	Runtime Statistics . . . . .	186



# List of Listings

---

7.1	Example of a Context Query . . . . .	89
8.1	Example of Metadata Constraints . . . . .	102
8.2	Example of Metadata Constraints 2 . . . . .	104
9.1	Example of Metadata Constraints 3 . . . . .	125
11.1	Demonstrator B: Logging Output of the Selection Service in the First Scenario . . . . .	166
A.1	Demonstrator B: Logging Output of the Selection Service for the Second Scenario . . . . .	197



# List of Algorithms

---

3.1	Resolution Algorithm . . . . .	33
3.2	Skolem Algorithm . . . . .	34
3.3	Gilmore's Algorithm . . . . .	35
3.4	Ground Resolution Algorithm . . . . .	35
8.1	Algorithm <i>checkMediation(query, chain<sub>in</sub>, level)</i> . . . . .	95
8.2	Algorithm <i>checkMetadataMediation(query, chain)</i> . . . . .	96
8.3	Algorithm <i>generateInputQueryForIRO(iro, offer)</i> . . . . .	99
8.4	Algorithm <i>generateOutputForIRO(iro, offer)</i> . . . . .	100
9.1	Algorithm <i>select(query, chainSet)</i> . . . . .	127





# List of Abbreviations

---

API .....	Application Programming Interface
AUF .....	Aggregated Utility Function
CAS .....	Context-Aware System
CMUF .....	Cost Minimization Utility Function
CNF .....	Conjunctive Normal Form
CoC .....	Cost of Context
COQL .....	Context Offering and Query Language
CQL .....	Context Query Language
DNF .....	Disjunctive Normal Form
FCL .....	Feedback Control Loop
GPS .....	Global Positioning System
IQR .....	Interquartile range
IRI .....	Internationalized Resource Identifier
IRO .....	Inter-Representation Operation
MOO .....	Multi-Objective Optimization
OWL .....	Web Ontology Language
QoC .....	Quality of Context
QoD .....	Quality of Device
QoS .....	Quality of Service
RDF .....	Resource Description Framework
RDF-S .....	Resource Description Framework Schema
SLA .....	Service-Level Agreement
SOC .....	Service Oriented Computing
SUF .....	Single Utility Function
UC .....	Ubiquitous Computing
URI .....	Uniform Resource Identifier
WGS84 .....	World Geodetic System 1984



# Abstract

---

In the vision of Mark Weiser on ubiquitous computing, computers are disappearing from the focus of the users and are seamlessly interacting with other computers and users in order to provide information and services. This shift of computers away from direct computer interaction requires another way of applications to interact without bothering the user. Context is the information which can be used to characterize the situation of persons, locations, or other objects relevant for the applications. Context-aware applications are capable of monitoring and exploiting knowledge about external operating conditions. These applications can adapt their behaviour based on the retrieved information and thus to replace (at least a certain amount) the missing user interactions. Context awareness can be assumed to be an important ingredient for applications in ubiquitous computing environments. However, context management in ubiquitous computing environments must reflect the specific characteristics of these environments, for example distribution, mobility, resource-constrained devices, and heterogeneity of context sources.

Modern mobile devices are equipped with fast processors, sufficient memory, and with several sensors, like Global Positioning System (GPS) sensor, light sensor, or accelerometer. Since many applications in ubiquitous computing environments can exploit context information for enhancing their service to the user, these devices are highly useful for context-aware applications in ubiquitous computing environments. Additionally, context reasoners and external context providers can be incorporated. It is possible that several context sensors, reasoners and context providers offer the same type of information. However, the information providers can differ in quality levels (e.g. accuracy), representations (e.g. position represented in coordinates and as an address) of the offered information, and costs (like battery consumption) for providing the information.

In order to simplify the development of context-aware applications, the developers should be able to transparently access context information without bothering with underlying context accessing techniques and distribution aspects. They should rather be able to express which kind of information they require, which quality criteria this information should fulfil, and how much the provision of this information should cost (not only monetary cost but also energy or performance usage). For this purpose, application developers as well as developers of context providers need a common language and vocabulary to specify which information they require respectively they provide. These descriptions respectively criteria have to be matched. For a matching of these descriptions, it is likely that a transformation of the provided information is needed to fulfil the criteria of the context-aware application. As it is possible that more than one provider fulfils the criteria, a selection process is required. In this process the system has to trade off the provided quality of context and required costs of the context provider against the quality of context requested by the context consumer. This selection allows to turn on context

sources only if required. Explicitly selecting context services and thereby dynamically activating and deactivating the local context provider has the advantage that also the resource consumption is reduced as especially unused context sensors are deactivated.

One promising solution is a middleware providing appropriate support in consideration of the principles of service-oriented computing like loose coupling, abstraction, reusability, or discoverability of context providers. This allows us to abstract context sensors, context reasoners and also external context providers as *context services*. In this thesis we present our solution consisting of a context model and ontology, a context offer and query language, a comprehensive matching and mediation process and a selection service. Especially the matching and mediation process and the selection service differ from the existing works. The matching and mediation process allows an autonomous establishment of mediation processes in order to transfer information from an offered representation into a requested representation. In difference to other approaches, the selection service selects not only a service for a service request, it rather selects a set of services in order to fulfil all requests which also facilitates the sharing of services. The approach is extensively reviewed regarding the different requirements and a set of demonstrators shows its usability in real-world scenarios.

# Zusammenfassung

---

In Mark Weisers Vision von ubiquitären Computern verschwinden Computer aus dem Blickfeld der Nutzer und interagieren nahtlos mit anderen Computern und Nutzern um Dienste und Informationen bereitzustellen. Durch das Verschieben des Computers in den Hintergrund des Nutzers wird eine andere Art an Anwendungen benötigt, um weiterhin zu interagieren allerdings ohne den Nutzer zu stören. Der Kontext einer Anwendung umfasst alle Informationen, die genutzt werden können um Situationen, Orte oder andere für die Anwendung relevante Objekte zu beschreiben. Kontextsensitive Anwendungen sind in der Lage ihre Umgebung zu überwachen und Wissen über externe Betriebsbedingungen anzusammeln. Diese Anwendungen können ihr Verhalten dynamisch an die gesammelten Informationen anpassen und somit (zumindest einen Teil) der fehlenden Nutzerinteraktion ersetzen. Kontextsensitivität kann also als wichtiger Baustein für Anwendungen im Umfeld von ubiquitären Computern angesehen werden. Dabei müssen kontextsensitive Anwendungen in Umgebungen ubiquitärer Systeme besondere Herausforderungen meistern, wie beispielsweise die Verteilung von Informationen über diverse Geräte, die Mobilität dieser Geräte, Beschränkung von Gerätere Ressourcen und Heterogenität von Informationsquellen.

Moderne mobile Geräte sind heutzutage mit schnellen Prozessoren, großem Speicher und mehreren Sensoren, wie Global-Positioning-System-Sensor (GPS), Lichtsensor oder Beschleunigungsmesser, ausgestattet. Da viele Anwendungen im Umfeld ubiquitärer Systeme Kontextinformationen nutzen, um ihre Dienste zu verbessern, eignen sich diese Geräte hervorragend um kontextsensitive Anwendungen auszuführen. Zusätzlich zu den Sensoren können so genannte Kontext-Reasoner und externe Informationsquellen, wie zum Beispiel ein in einem Gebäude installiertes Lokalisierungssystem existieren und somit auch als weitere Informationsquellen auf dem mobilen Gerät genutzt werden. Dabei ist es möglich, dass all diese Sensoren und andere Quellen Informationen des gleichen Typs zur Verfügung stellen. Allerdings können sich diese Dienste in Qualitätsstufen (z. B. Genauigkeit), Darstellungsform der angebotenen Informationen (z.B. Position in Koordinaten und als Adresse dargestellt) und in den Kosten für die Bereitstellung der Informationen unterscheiden (z.B. Batterieverbrauch).

Um die Entwicklung von kontextsensitiven selbstadaptiven Anwendungen zu erleichtern, sollten die Entwickler in der Lage sein, transparent auf Kontextinformationen zuzugreifen, ohne sich mit den zugrunde liegenden Techniken bezüglich Zugriff und Verteilung zu beschäftigen. Sie sollten lediglich ausdrücken müssen, welche Art von Information sie benötigen und welche Qualitätskriterien und Kostenlimits bei der Bereitstellung eingehalten werden sollen. Daher benötigen sowohl Anwendungsentwickler als auch Entwickler von Informationsquellen eine gemeinsame Sprache und Vokabular um genau angeben zu können, welche Information sie bereitstellen bzw. benötigen. Diese Beschreibungen müssen anschließend auf Übereinstimmungen überprüft werden. Dabei ist es möglich, dass Informationen zunächst noch transformiert werden müssen,

bevor sie den Kriterien des Konsumenten entsprechen. Da es zudem möglich ist, dass mehrere Anbieter zur Verfügung stehen, die die Kriterien erfüllen, wird ein Auswahlverfahren benötigt um den besten Anbieter zu finden. Dabei muss das System die bereitgestellten Qualitätsmerkmale und benötigten Kosten der Kontextquelle gegen die vom Kontextverbraucher erhobenen Anforderungen bezüglich Kosten und Qualität abwägen. Dieses Auswahlverfahren erlaubt es auch, Informationsquellen nur zu aktivieren, wenn diese auch wirklich benötigt werden. Diese explizite Aktivierung bzw. Deaktivierung von lokalen Kontextanbietern reduziert den Ressourcenverbrauch enorm.

Eine viel versprechende Lösung ist eine Middleware, die geeignete Unterstützung bereitstellt und dabei die Prinzipien des Service-Oriented Computing wie lose Kopplung, Abstraktion oder Wiederverwendbarkeit von Kontext-Anbietern berücksichtigt. Dies erlaubt uns, Sensoren, Kontext-Reasonern und auch externen Anbietern zu Kontextdienste zu abstrahieren. In dieser Arbeit präsentieren wir unsere Lösung, bestehend aus einem Kontextmodell und einer Ontologie, einer Sprache um Kontextangebote bzw. -anforderungen zu spezifizieren, einem umfassenden Matching- und Mediationsverfahren und einem Auswahlverfahren für geeignete Kontextdienste. Insbesondere das Matching- und Mediationsverfahren und das Auswahlverfahren heben sich von den bereits existierenden Ansätzen ab. Das Matching- und Mediationsverfahren ermöglicht die autonome Erstellung von Ketten sogenannter Kontextvermittler um Informationen aus einer angebotenen Form in die angeforderte Form zu überführen. Der Dienst zur Auswahl von Kontextquellen wählt im Unterschied zu anderen Ansätzen nicht nur einen Service für eine Serviceanfrage, vielmehr wählt er eine Reihe von Diensten um damit alle Anfragen zu erfüllen. Dies erlaubt auch die gemeinsame Nutzung von Kontextquellen durch mehrere Konsumenten. Der präsentierte Ansatz wird ausführlich diskutiert und insbesondere in Bezug auf die unterschiedlichen Anforderungen evaluiert. Dabei zeigen mehrere Demonstratoren die Verwendbarkeit des Ansatzes in realen Szenarios.

# Acknowledgements

---

*Never lose the child-like wonder.  
It's just too important.  
It's what drives us.*

– Randy Pausch (1960-2008)  
The Last Lecture (2007)

Writing this thesis would have never been possible without the help, the discussions, the criticisms and last but not least the motivations by so many people. Even if I am not a man of many words, I have to thank everyone who has tenaciously asked for the current status of my work and has gotten more than once the short answer “Little by little, the bird builds its nest (Mühsam ernährt sich das Eichhörnchen)”.

First and foremost I want to thank my advisor *Kurt Geihs*, who provided his support and guidance throughout the progress of my dissertation. Furthermore I also would like to thank my second advisor *Christian Becker* for the fruitful discussions before writing this document. A thank you also goes to my recently appointed committee members *Klaus David* and *Arno Wacker*.

Working in Kurt's department at the University of Kassel was really a pleasure as I was able to work with a lot of excellent, friendly and helpful researchers, colleagues and friends. Here, I have to highlight *Roland Reichle* as without his help this thesis would have never been finished. Thanks for all the discussions and motivations. Furthermore I have to thank *Christoph Evers*, *Philipp Baer*, *Michael Zapf*, and *Eva Mellom* for proofreading this thesis. However, I do not want to forget to mention the rest of my colleagues: *Till Amma*, *Steffen Bleul*, *Diana Comes*, *Mohammad Ullah 'Titu' Khan*, *Dominik Kirchner*, *Thomas Kleppe*, *Alex Kohout*, *Stefan Niemczyk*, *Iris Roßbach*, *Daniel Saur*, *Hendrik Skubch*, *Thomas Weise*, and *Andreas Witsch*. Thank you for all the discussions and inspirations.

During an important phase of my short life as a researcher, I have been part of the *MUSIC project consortium*. It was a great pleasure to work with so many inspiring and experienced researchers. I would especially like to thank *Nearchos Paspallis* from the University of Cyprus for his excellent work and the valuable discussions with him. In addition, I want to emphasize *Geir Horn*, *Svein Hallsteinsen*, *Erlend Stav*, *Jacqueline Floch* (all SINTEF), *Frank Eliassen* (University of Oslo) and *Romain Rouvoy* (University Lille 1).

Finally, I have to thank my beloved wife, my little son, my parents, my siblings, my parents-in-law, the rest of my family and also my friends. Thank you so much for being extremely patient with me!





**Part I**

**Foundations**



# 1 Introduction

---

*In writing a problem down or airing it in conversation we let its essential aspects emerge. And by knowing its character, we remove, if not the problem itself, then its secondary, aggravating characteristics: confusion, displacement, surprise.*

– Alain de Botton (1969-)  
The Consolations of Philosophy, 2000

## 1.1 Motivation

Following the vision of Mark Weiser, ubiquitous computing is getting more and more important these days [145]. In this vision, computers are disappearing from the focus of the user and are seamlessly interacting with other computers and users in order to provide information and services. Weiser's vision becomes a reality due to the substantial progresses in the different computational technologies. Size and powerfulness of computing devices are growing inversely proportional. This means devices are getting smaller, but are also equipped with faster and more powerful processors, more memory, and also with a lot of sensors. Nevertheless, devices and their equipment are also becoming very energy efficient and are able to communicate using several different communication standards.

The movement of devices into the background of the user does not only require a significant progress in the development of hardware but it also requires a change of software development paradigms. The user should be bothered as less as possible. Applications have to work autonomously based on information they retrieve from different sources (like sensors, databases, or also other devices and applications). These applications are called context-aware and self-adaptive. In recent years, context awareness (see Definition 2.1 and Definition 2.2) has attracted a lot of attention, especially in the realms of mobile and ubiquitous computing. Context-aware applications are capable of monitoring their surrounding environment and exploiting knowledge about external operational conditions.

In addition to context awareness, applications in ubiquitous computing environments have not only to collect information but they are also required to react on changing information. Software that dynamically adapts as a response to changes in the execution context is called self-adaptive software. As stated by Geihs, self-adaptive software in general and not only in ubiquitous computing environments needs to be context-aware, too [41].

In ubiquitous computing environments, applications have to struggle with additional challenges supplementary to the requisite to be context-aware and self-adaptive. Schiele et al. mention dynamic discovery, data interpretation, and energy-saving as essential requirements in ubiquitous computing [120]. Context management in ubiquitous computing environments must reflect the specific characteristics of these environments, for example distribution, mobility, resource-constrained devices, and heterogeneity of context sources.

Modern mobile devices are equipped with fast processors, plenty of memory space, and several sensors, like a Global Positioning System (GPS) sensor, a light sensor, or an accelerometer. Since many applications in ubiquitous computing environments can exploit context information for enhancing their service to the user, these devices are highly useful for context-aware applications in ubiquitous computing environments. Additionally to sensors, context reasoners can also be deployed on these devices and can retrieve new context information from other (context) information. External context providers, like a localization system installed in a building or a shared information source provided by another device, can also be incorporated. It is possible that several context sensors, reasoners and other context providers offer the same type of information. For example, a modern smartphone can retrieve the location via a GPS sensor or it can use a WiFi-based localization. Besides, an external location service may be available in a building.

However, as stated by Becker, “[. . .] if multiple applications and context sources feed their data into a context model multiple representations of an object may exist. The context model has to provide concepts to deal with such phenomena, e.g., choosing one representation, combining them, or prompting the user.” [9]. Becker et al. provide also concrete examples: “Information about locations is presented in different formats. Geometric coordinates as they are used by GPS refer to a point or geometric figure in a multi-dimensional space, typically a plane or a three-dimensional space. [. . .] Symbolic coordinates on the other hand do not provide any reasoning about their spatial properties (distance and inclusion) without any additional information. Such coordinates are available via cell-IDs in cellular networks, such as GSM or wireless LAN, as well as via other positioning technologies, such as radio frequency tags (RFIDs) or infrared (IR) beacons.” [10].

More generally, context services can differ in quality levels (e.g. accuracy), representations (e.g. position represented in coordinates or as an address) of the offered information, and costs (e.g. battery consumption) for providing the information. For example a context service encapsulating a GPS sensor provides information regarding the current device position in form of WGS84 coordinates with a high accuracy but consumes a lot of energy. In difference, another context service retrieving the current position from calendar entries could provide the current position in form of an address without requiring much additional resources. However this address can deviate in its accuracy.

In order to simplify the development of context-aware self-adaptive applications, the developer should be able to access context information transparently without bothering with underlying context access techniques, distribution aspects, and the additional challenges caused by the ubiquitous computing environment. As stated by Lehmann et al., “The variety of possible information sources and services, such as sensors, actuators, and user interfaces, should be transparent to applications. A layer of abstraction should be

*provided to facilitate the easy change of resources, such as upgrading a positioning system. Newly integrated resources should be made available to applications.*” [9, 82]. Instead of accessing a certain context provider, developers should rather be able to express which kind of information they require, which quality criteria this information should fulfil, and how much the provision of this information should cost (not only monetary cost but also energy or performance usage).

For this purpose, the application developer as well as the developer of context providers need a common language and vocabulary to specify which information they require respectively they provide. Becker states that “[. . .] *the interpretation of context data across applications requires a common semantic. Examples are a common type schema or ontology.*” [9]. These descriptions respectively criteria have to be matched. For this matching, it might be possible that a transformation of the provided information is needed to fulfil the criteria of the context-aware application. As it is possible that more than one provider matches the criteria, a selection process is required. For the selection, the system has to trade off the provided quality of context and required costs of the context provider against the quality of context requested by the context consumer.

As already stated by Wang et al., “*continuously capturing this contextual information on mobile devices consumes huge amount of energy*” [144]. Especially on mobile devices, it is important to reduce the energy consumption. In addition to costs for measuring or calculating a piece of context information, which describe some kind of resource consumption, it is also desirable to reduce other costs like monetary expenses. Hence, context services should only be activated when selected and otherwise they should be deactivated. This activation & deactivation strategy saves a lot of costs but also causes some problems. For example, context services have to provide a detailed description of the provided information especially of the metadata as no control sample can be received from the service to check whether a service provides the information as requested or not.

One promising solution is a middleware providing appropriate support in consideration of the principles of service oriented computing (SOC) like loose coupling, abstraction, reusability, or discoverability of context providers [38]. This allows us to abstract context sensors, context reasoners and also external context providers as *context services*. In this thesis, we present our solution consisting of a context model and ontology, a context offer and query language, a comprehensive matching and mediation process and a selection service. Especially the matching and mediation process and the selection service differ from the existing works. The matching and mediation process allows an autonomous establishment of mediation processes in order to transfer information from an offered representation into a requested representation.

Similar to context services, where several services exist that only differ in the quality or cost of the provided information, context-aware applications can also have diverse requirements regarding the information. A middleware for developing context-aware self-adaptive applications has to provide support for selecting appropriate context services. The middleware also has to trade off the provided quality of context and required costs of the context service against the quality of context requested by the context consumer. Selection approaches in related works typically search for a service sequentially request by request. In contrast, our selection service selects a set of services in order to fulfil all requests which also facilitates the sharing of services. This selection allows to turn on context sources only if required. Explicit selection of context services and thereby

dynamically activating and deactivating the local context provider has the advantage that also the resource consumption is reduced as especially unused context sensors are deactivated.

The multi-service selection and the dynamic activation & deactivation are the main reasons why existing concepts for service selection like the approaches by Jaeger [62] or by Yang et al. [148] are not appropriate for selecting context services (see related work discussion in section Chapter 4 and Chapter 9).

## 1.2 Problem Statement

The general objective of this work is a comprehensive approach allowing the abstraction of heterogeneous context providers as context services. In this approach, the previously motivated challenges like the selection and integration of dynamically discovered context sources and exchanging the provided, heterogeneously represented context information in a ubiquitous computing environment have to be considered. These context services can be dynamically discovered, selected, and activated based on the requirements of a context consumer. In all phases from the discovery until the selection, the matching of offered and required context information and the metadata, like the provided quality levels and costs have to be taken into account.

The *Mobile Users in Ubiquitous Computing Environments* (MUSIC)<sup>1</sup> platform [96, 151] and the PhD theses of Reichle [109] and Paspallis [104] serve as a basis for this work. MUSIC offers a middleware and methodology for the development of self-adaptive context-aware applications in ubiquitous computing environments and explores advanced compositional adaptation by considering dynamically discovered services as possible replacements for application components. Furthermore, the platform offers extensible support for accessing context data. Context accessing methods are encapsulated into context plugins which can be transparently accessed by context consumers with a context query language. Paspallis' work forms a central part of the MUSIC context middleware and hence serves also as a basis for this work. Reichle also builds on this middleware and significantly extends it by supporting heterogeneously represented context information and information fusion.

We identified the following requirements that form the work plan for this thesis and that are not or only partially addressed in MUSIC or by Paspallis or Reichle:

**Requirement 1: Semantic discovery and integration of independently developed context services and consumers.** In an ubiquitous computing environment, the appearance and disappearance of users, their devices, and services hosted on these devices have to be considered. This has several reasons. For example, it can not be assumed, that users and their devices are permanently available (devices may be turned off or have no network connection). In addition, it is impossible to know all devices and services in a ubiquitous computing environment at design-time. For that reason, run-time mechanisms are required to discover context services and context consumers, to reason about, and to perform the mediation tasks that are needed to bridge the heterogeneity issues arising from the independent development of the involved context services.

---

<sup>1</sup>MUSIC: Mobile Users in Ubiquitous Computing Environments, IST FP6 IP 035166, <http://ist-music.berlios.de>. Last visited on Jan 12, 2012.

**Requirement 2: Loose coupling of context providers and consumers.** Context services dynamically appear and disappear, runtime conditions of services may influence the quality of the provided information, and the requirements regarding context information and their metadata can change. So, applications should be able to switch dynamically between different context services and context services should not be hard-wired in the source code. The principle of loose coupling<sup>2</sup> promotes the independent design and evolution of a context service's logic and implementation. This further allows to compose the different components encapsulating context provisioning mechanisms to so-called context reasoners. These context reasoners retrieve higher-level context information by composing, aggregating, or interpreting low-level context information of one or more context sources.

**Requirement 3: Exchange and interpretation of heterogeneously represented context information.** Independent development of context providers and consumers implies that each development team utilizes the most suitable platform and technology for its task, but also it names and represents the data and metadata according to its needs. Even if platform-independent data exchange formats like XML are used and if context providers try to satisfy the desires of the consumers, the independent development results in naming conflicts and in heterogeneous representations of data and metadata. For example, the location of the user in a ubiquitous computing environment may be given in GPS coordinates or as room number of a building. Additionally, the accuracy of the location may be specified as radius around that position in millimeter or as value ranges for the coordinates. A common vocabulary has to be defined that allows to semantically interpret the meaning and representation of the data and metadata. This is a prerequisite to reason about the needed mediation tasks to achieve interoperability at runtime when exchanging context information but also at selection time when selecting a certain context service based on the metadata. In particular, this also comprises the conversion between different data representations, as for example the conversion of a room number to GPS coordinates.

**Requirement 4: Expressing context offers and needs.** Krause et al. assume that context-aware services should not have hard-linked context sensors but have to search for context providers at runtime due to the increasing mobility of users and devices [75]. In order to establish communication links between context consumers and providers in a dynamic fashion, also context offers and needs have to be expressed based on a common vocabulary as mentioned in Requirement 3. A language should allow the elaborate specification of context needs, to filter out inappropriate context offers and to establish only communication links that provide the information actually needed.

**Requirement 5: Activation and deactivation of local context services.** Mobile devices are limited in their capabilities like CPU and also in their resources like battery or memory space. Hence, software on a mobile device should be as energy efficient as possible. Especially sensors are known to be very resource consuming. In order to minimize the resource consumption, local context providers should be activated only when required and unused providers should be deactivated.

**Requirement 6: Dynamic selection of context services based on quality and cost attributes.** The set of available context services providing the same type of context information varies over time. Context services appear and disappear and the quality

---

<sup>2</sup>The principle of loose coupling is also adopted from the SOC principles

levels, representations, and costs for providing the information vary between the different providers and also over time. Similar, the set of (local and remote) context consumers and their requirements regarding context information, their quality attributes, and the costs may vary. A mechanism is required for selecting an appropriate set of context services while taking into account provided and required quality levels, costs for the calculation, the consumer's preferences regarding costs, and the heterogeneous representations. The selection algorithm additionally has to address the challenge of activation and deactivation of context services. The activation & deactivation of context services result in using average and potentially not up-to-date quality data for the selection. This can have several consequences. For instance, after selecting and activating a provider, its actual quality values can be much worse than the predefined quality and also worse than the quality levels of the second-choice provider. This would result in a deactivation, a new run of the selection algorithm and potentially in the selection of the same context provider.

**Requirement 7: Minimizing total amount of resources used by context services.** While the selection of context services, it has to be considered that several context consumers can request the same type of context information with slightly different requirements. In order to minimize the resource consumption, appropriate support has to be provided to minimize the number of selected respectively activated context services for example by sharing context services. In addition, the cheapest context service should be selected which satisfies the requirements of a context consumer with respect to the required quality of context.

### 1.3 Solution Approach

Figure 1.1 provides an overview of the proposed approach. The baseline of the overall approach is a context model and an ontology which define the semantic concepts of *Entity types*, *Scopes* and *Representations*. An entity is a physical or logical entity of the world that is described by the context information, e.g. a smartphone or a person. The scope refers to the type of the provided information, e.g. the location. Metadata are also considered as scopes and finally a representation describes how the context information is internally structured. The context ontology is used to provide a common vocabulary to bridge semantic differences by defining the semantic concepts for entity (concrete individuals) respectively entity types (classes of individuals), scopes and representations. In addition, the ontology captures the relationships between the defined concepts. The internal organization of context information and their quality attributes are defined as representations in the ontology. By providing Inter-Representation Operations (IROs) similar to Strang et al. [134], we allow the conversion between different representations. With the help of these concepts a common vocabulary is established that enables interpreting the meaning and representation of the exchanged data.

As highlighted in the previous section, several context services and consumers can exist in parallel. The required and offered context information are specified with the *Context Offer and Query Language (COQL)* and referencing the common vocabulary defined by the ontology. The COQL makes it possible to precisely define context offers and request. The corresponding semantic definitions serve as input for the *Discovery and Matching* approach. Matching of context offers and requests already includes the reasoning on



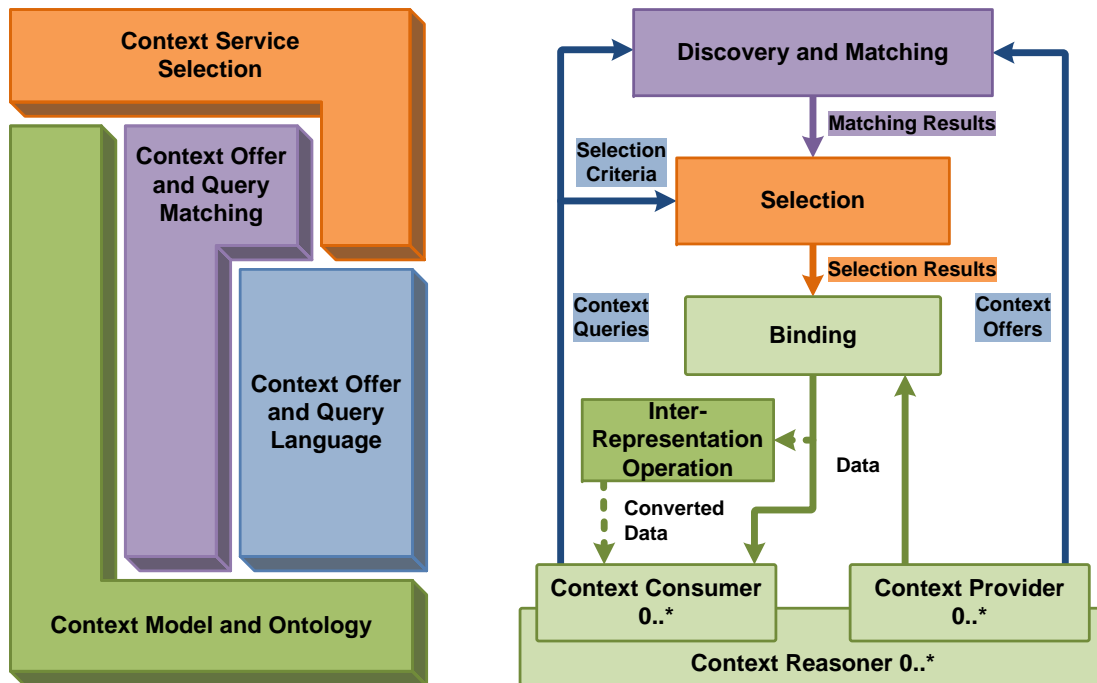


Figure 1.1: Overview of the Solution Approach.

potentially required mediation tasks in form of IROs to overcome mismatches in the provided and required representations of the context information. Potentially required IROs to mediate between the representations of selection criteria (context metadata in general) have to be appointed to ensure comparability of offers and requests in the selection phase.

The results of the *Discovery and Matching* serve as input for the *Selection* step. Before the actual selection, the discovery and matching results have to be analysed for temporal infeasible context services. As mentioned in the previous section, the quality information of a context service does not necessarily have to reflect the actual quality, as the service might currently be deactivated. This results in using estimated and potentially not up-to-date quality data for the selection. Unfeasible context services are removed from the selection until a significant context change has happened. After a significant context change, a correspondence of estimated quality attributes and actual quality attributes after activation is more likely. For the actual selection, a combined utility function is used to calculate the utility of a set of context services regarding all registered context requests. This combined utility functions aggregates the selection functions expressed by the different context requests in consideration of the provided and required quality and cost data. The set of context services with the highest utility is selected. A context service can also be shared by several context requests. If a context service does not provide the required representation but an appropriate IRO is available (already checked in the *Discovery and Matching*), the costs of the IRO also have to be taken into account. Furthermore an IRO can result in a change of the quality of the provided information. The result of the *Selection* is used to establish communication links between context provider and consumer. The selected context services will be activated and optionally

context mediator chains consisting of  $1 \dots n$  IROs will be established to provide the requested representation of the context information.

In summary, our solution provides support for the dynamic discovery and selection of context services based on a matching of context offers and context requests. Context providers (sensors and reasoners) are encapsulated as loosely coupled context services. Our system is able to overcome heterogeneously represented context and metadata, offered by these context services, by transferring them in comparable representations. In addition this approach provides support to handle the challenges introduced by the functionality of dynamically activating and deactivating context services.

## 1.4 Contribution

The research work and results documented in this thesis lead to a number of contributions to the state-of-the-art research. Most of the contributions are made in the area of *Context Management*, *Context Modelling*, *Self-Adaptation* but also *SOA* and *Web Services in Ubiquitous Computing Environments*. We provide the following major contributions:

1. a **comprehensive context model and ontology** combining existing approaches in order to meet all the requirements described in Section 1.2 [158],
2. a **Context Query language** building the baseline for our *Context Offering and Query Language* [159],
3. an approach for the **dynamic integration of services** into a component-based self-adaptive application [154, 152],
4. the **general solution approach** described in this thesis, which has been iteratively developed within the last years [165, 168],
5. the **matching of context offers and queries** along with the autonomous transformation of heterogeneously represented context information,
6. and the **selection of context services** under consideration of constraints caused by the dynamic activation and deactivation of context providers.

## 1.5 Structure of the Thesis

The document is divided in three parts. The first part describes the foundations and existing works, which serve as the basis for the rest of the document. The second part describes our solution approach in detail, while the last part evaluates the solution and discusses its advantages and disadvantages. Part I has four chapters, Part II six, and Part III is divided into three chapters. The content of the rest of the document is summarized as follows:

**Chapter 2:** In this chapter, the general terms like *ubiquitous computing* or *context awareness* are defined.

**Chapter 3:** Checking satisfiability of logical expressions builds an important basis for the matching of context offers and queries. In this chapter, all required logical foundations are introduced. Furthermore, a short introduction to ontologies is provided.

**Chapter 4:** Based on the requirements for our work introduced in Section 1.2, existing works are evaluated and discussed in this chapter.

**Chapter 5:** This chapter gives a short overview of our solution approach and introduces the relevant stakeholders for the development of context-aware applications following our approach.

**Chapter 6:** The context model builds the semantic basis of our system. In Chapter 6 we introduce the context model and associated concepts like the mediation operators.

**Chapter 7:** In order to support the loose coupling and discoverability, both, context consumers and context providers need a way to describe which information they need respectively they provide. For these descriptions, we developed the *Context Offer and Query Language*, which is described in this part of the thesis.

**Chapter 8:** Context offers and requests have to be evaluated to decide which offer potentially can provide the requested information. Furthermore it might be possible that mediation is required to transform information into a specific representation.

**Chapter 9:** Several context providers may offer the same kind of information. Hence, a selection is required to decide which providers to activate.

**Chapter 10:** In this chapter, the resulting architecture and implementation details of our middleware are discussed.

**Chapter 11:** As a basis for our demonstrators, we have done a study on context information, QoC and CoC used in existing works. This study is used as an input for implementing a set of context services, which are used afterwards in the different demonstrators. The demonstrators show the fulfilment of the different requirements and the feasibility of our approach.

**Chapter 12:** In the different discussion sections of the chapters describing our solution approach, the computation complexity of our solution has been discussed from the theoretical point of view. In Chapter 12, performance and scalability is evaluated in practise and the results are compared to the theoretical results.

**Chapter 13:** Finally, this chapter concludes this thesis and presents an outlook on future work.



## 2 Ubiquitous Computing Systems

---

*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*

– Mark Weiser (1952-1999)  
Scientific American, September 1999

With the appearance and penetration of mobile devices such as notebooks, tablet computers and smartphones, ubiquitous computing systems are becoming increasingly popular. The term ‘ubiquitous computing’ introduced first by Weiser refers to the seamless integration of devices into the user’s everyday life [145].

As indicated by Langheinrich et al., Weiser used the term ‘ubiquitous computing’ and associated it with an idealistic and human-centric technology vision, which can only be realized in the distant future [80]. With a more pragmatic emphasis the industry in particular IBM, has coined the term ‘pervasive computing’. Similar to ubiquitous computing, it is about the omnipresent information processing, however, with the primary goal of using existing mobile computing technologies, which are already available in short term. In addition to these two terms, the term ‘ambient intelligence’ emerged including aspects of the human-machine-interaction and artificial intelligence. However as stated by Langheinrich et al., the distinction between the three concepts is rather academic and the similarities predominate [80]. For this reason, we use the terms ‘ubiquitous computing’, ‘pervasive computing’, and ‘ambient intelligence’ interchangeable in the rest of this thesis.

As terms like context awareness and self-adaptation are inevitably connected with ubiquitous computing (UC) and as these concepts are the most important properties of UC for this work, we first define UC in general and then focus on the specific aspects of context like definition of context and context awareness and characteristics of context data in Section 2.2 and afterwards on self-adaptation in Section 2.3.

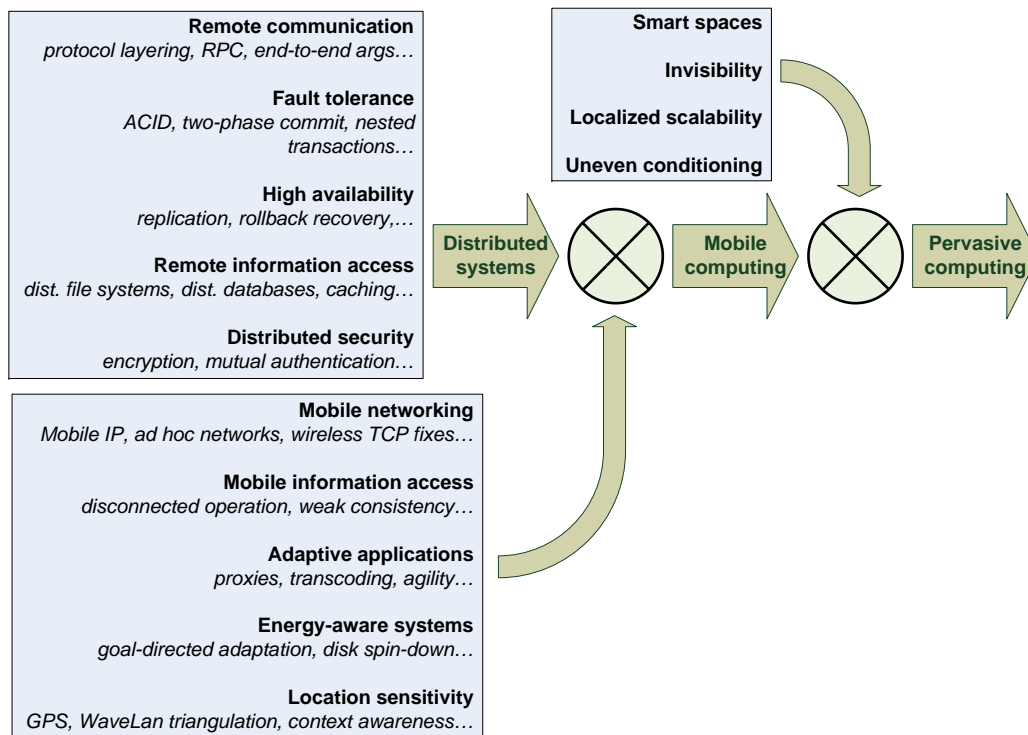
### 2.1 Ubiquitous Computing

In the dictionary of *reference.com* the term ubiquitous is defined as follows:  
**u·biq·ui·tous** [yoo·bik·wi·tuhs] – adjective – existing or being everywhere, especially at the same time; omnipresent: ubiquitous fog; ubiquitous little ants<sup>1</sup>.

<sup>1</sup><http://dictionary.reference.com/browse/ubiquitous>, last visited on Nov. 10, 2011.

In difference to other topics, where it is very hard (if not impossible) to state the first person mentioning the topic, *Ubiquitous Computing* is inevitably associated with Mark Weiser. He states “*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*” [145].

Satyanarayanan explains the vision of Weiser regarding Ubiquitous Computing and provides a rather formal explanation [119]. According to him, UC is based on the wide research topics of *distributed systems* and *mobile computing*. Satyanarayanan provides a taxonomy of computer research problems in pervasive computing, which is depicted in Figure 2.1.



**Figure 2.1:** Taxonomy of Computer Systems Research Problems in Pervasive Computing [119]

As indicated by the modulation symbol, Satyanarayanan notes that pervasive computing takes advantages of distributed and mobile computing while inheriting problems in these fields increasingly. Thereby also the combination of distributed and mobile computing solves not all problems of UC. Lytinen et al. share this opinion. They state that “*In mobile computing, however, an important limitation, is that the computing model does not considerably change while we move. This is because the computing device cannot seamlessly and flexibly obtain information about the context in which the computing takes place and adjust it accordingly.*” [84].

In the literature a wide range of different challenges or requirements have been discussed that have to be solved in order to provide software respectively systems which are “UC-ready”. Here, we only give an overview of the most important ones (the interception set of all challenges mentioned in [52, 67, 84, 98, 116, 119, 129]). It is possible to divide this set of challenges into two: a list of underlying properties/technologies that serve as a baseline of UC systems and a set of cross-cutting aspects/challenges that have to be handled within all items of the first list.

### Foundations of ubiquitous computing:

- **Context awareness:** A central aspect of ubiquitous computing systems is mobility. Hence, runtime conditions of applications/systems change continuously. However, as mentioned by Weiser, technology in UC systems should disappear [145]. To accomplish this invisibility, the user interaction with the system has to be reduced whereas the system has to react on several more aspects as standard desktop applications. Thus, the system requires new ways to retrieve the required information. For this reason, the ubiquitous computing systems have to be context-aware. The different aspects of context awareness are explained in detail in Section 2.2.
- **Self-Adaptation:** Context awareness alone is not sufficient to replace the reduced user interaction and to satisfy the vision of disappearing technology. The system has to react autonomously on the changing conditions and thus on the retrieved context information. Therefore, it has several options, like changing parameters, adding/exchanging/removing components or binding external services. This aspect is called self-adaptation and is described in detail in Section 2.3.

### Cross-cutting concerns in ubiquitous computing:

- **Heterogeneity:** Within a UC environment, many different devices, software fragments, and user interfaces exist, which makes heterogeneity to a central and also cross-cutting challenge in UC systems. To highlight the importance of this aspect, Saha et al. propagate “*Middleware must mask heterogeneity to make pervasive computing invisible to users.*” [116]. Henricksen et al. therefore note “[...] *heterogeneous devices will be required to interact seamlessly, despite wide differences in hardware and software capabilities.*” [52]. As stated by Niemelä et al., “[...] *heterogeneity of software is expressed by a diversity of software structures, component models, interface technologies and languages.*” [98]. “*This will require dynamic interoperability at the component level, in addition to interoperability that overcomes the heterogeneity of the environment and of components.*” as mentioned by Henricksen et al. [52].
- **Invisibility:** Weiser et al. state “Ubiquitous computing will require a new approach to fitting technology to our lives, an approach we call ‘calm technology.’” [146]. In the vision of calm computing, human intervention with the system is reduced to a minimum and hence the system becomes nearly invisible. This central point has already been used to motivate *context awareness* and *self-adaptation*.
- **Masking uneven conditions:** This aspect is introduced by Satyanarayanan, who states that it is not very likely that a uniform penetration of pervasive computing technologies with different environments will be reached in the next decades [119]. For this reason, a UC system has to be able to work in both extremes: in environments fully penetrated with technology and in environments without any UC technology. Besides, the system has to be able to detect changes in the environments also with regard to its equipment (→ context awareness) and has to adapt appropriately to this changes (→ self-adaptation).
- **Scalability:** UC systems may consist of thousands of devices, sensors, services, and users. Hence, scalability is an aspect that influences all parts of such systems.

- **Privacy and security:** As aforementioned, user activities should be reduced in order to increase the invisibility of UC actors. But the central ingredients of UC, context awareness and self-adaptation, open a wide range of potential security and privacy issues, for example it has to be proven to which actor context information is revealed and if the communication between context provider and consumer is secure.

## 2.2 Context

Context (from Latin *contextus* connection of words, coherence, from *contexere* to weave together, from *com-* + *texere* to weave<sup>2</sup>) in general describes the conditions and circumstances of an entity.

Various different definitions of the term *context* exist in computer science literature. Schilit et al. state that “[...] *context encompasses more than just the user’s location, because other things of interest are also mobile and changing. Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situation; e.g., whether you are with your manager or with a co-worker.*” [121]. They define context as “[...] *where you are, who you are with, and what resources are nearby.*”

Gwizdka distinguishes between internal and external context [49]. Internal context is the state of the user (composition of work context, personal events, communication context, and emotional state of the user etc.) whereas external context describes the state of the environment (composition of location, proximity of other people or devices, and temporal context etc.).

One of the most accurate (and widely referenced) definitions of context is given by Abowd et al. [2]. They refer to context as “*any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves.*”

Becker et al. adopt and revise the definition on the definition of Abowd et al. [11]. We adopt this definition and define context as follows:

**Definition 2.1** (Context). Context is the information, which can be used to characterize the situation of an entity. Entities are persons, locations, or objects which are considered to be relevant for the behaviour of an application. The entity itself is regarded as part of its context.

### 2.2.1 Context Awareness

Schilit et al. were one of the first who mentioned context awareness: “*context-aware software adapts according to the location of the user, the collection of the nearby people, hosts, and accessible devices, as well as to changes to such things over time.*” [121]. However, this was rather a collection of examples than a concrete definition.

<sup>2</sup>See <http://www.merriam-webster.com/dictionary/context>. Last visited Apr. 10, 2012.



Abowd et al. define context awareness as follows: “A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.” [2].

Similarly, Chen et al. state that “context-awareness [...] can be defined as a computer system’s ability to provide relevant services and information to users based their situational conditions.” [20].

Whereas the definitions by Abowd et al. and Chen et al. focus on the provisioning of information and/or services to the user, other authors specify more general definitions. For example, Razzaque et al. define context awareness as a “[...] term from computer science, which is used for devices that have information about the circumstances under which they operate and can react accordingly.” [108].

Becker et al. state that “an application is context-aware if it adapts its behavior depending on the context.” [11].

Baldauf et al. introduce context-ware systems as systems that “[...] are able to adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account.” [8].

More concisely, Huebscher et al. write “Context-awareness is the ability of an application to adapt itself to the context of its user(s).” [59].

The last two definitions highlight the context of the user. As we think, both – the context of the user of the application and the context of the application itself – is important, the following definition will be used in this thesis:

**Definition 2.2** (Context Awareness). An application is context-aware, if its behaviour is influenced by information on its context.

### 2.2.2 Characteristics of Context Data

Henricksen et al. discuss in their paper the different characteristics of context information [53]. This discussion can be summarised as follows:

- “Context information exhibits a range of temporal characteristics. Context information can be characterized as static or dynamic. [...] The persistence of dynamic context information is highly variable;”
- “Context information is imperfect. [...] Information may be incorrect if it fails to reflect the true state of the world it models, inconsistent if it contains contradictory information, or incomplete if some aspects of the context are not known.”
- “Context has many alternative representations. [...] For example, a location sensor may supply raw coordinates, whereas an application might be interested in the identity of the building or room a user is in.”
- “Context information is highly interrelated.”

After Dey et al. “both sensed and interpreted context is often ambiguous. A challenge facing the development of realistic and deployable context-aware services [...] is the ability to handle ambiguous context” [34].

The different reasons for unreliable or error-prone context information are discussed by Krause et al. [75]:

- Necessary context information (or context sources) might not be available.
- Context information might be out-dated and no longer applicable to the current situation.
- Physical constraints and temporary effects limit the precision of the sensors.
- Reasoning rules which often base on probabilities do not apply in certain situations. As a result, they could derive false context information.
- Information from malicious extrinsic sources could feign a context that is not real.

Motivated by these characteristics, Quality of Context is an central aspect in this thesis.

### 2.2.3 Quality of Context

As one of the first, Buchholz et al. introduced a new set of quality parameters for context information [17]. The quality of context information (precision, probability of correctness, trustworthiness, resolution, and up-to-date[d]ness<sup>3</sup>) is according to them neither identical to Quality of Service (QoS), nor to the quality of the underlying hardware components, i.e. Quality of Device (QoD). This new set has been called Quality of Context (QoC):

*“Quality of Context (QoC) is any information that describes the quality of information that is used as context information. Thus, QoC refers to information and not to the process nor the hardware component that possibly provide the information.”* [17].

Buchholz et al. identified precision, probability of correctness, trust-worthiness, resolution and up-to-dateness as the most important QoS-parameters. Besides, they have pointed out the differences between QoC, QoS and QoD. While QoC describes the quality of information, QoS refers to the quality of a service and QoD is any information about a device’s technical properties and capabilities. QoC is not equal to QoS and QoD, since context information can exist without services and devices. Nevertheless, QoC, QoS and QoD influence each other. As depicted in Figure 2.2, Buchholz et al. distinguish between two possibilities in which one notion of quality can affect another one:

- In the **bottom-up approach**, impacts of one quality on another are indicated through arrows that point upwards. Relating to QoC, the device dependent capabilities (QoD), which can also deviate, influence the QoC. Similar QoS influences QoC.
- In the **top-down approach**, qualities influence each other via the requirements they pose.

Buchholz et al. introduce also several examples for the usage of QoC:

- QoC agreements: *“When several actors cooperate to provide CASs [context-aware systems] [...], there is a need for contracts that not only specify the required QoS, but also address the QoC. In analogy to SLAs [service-level agreements <sup>4</sup>], we call contracts defining QoC requirements QoC agreements.”* [17].

<sup>3</sup>The terms “up-to-dateness” and “up-to-datedness” are used interchangeable.

<sup>4</sup>According to Erl, a SLA is “[...] a document that establishes a contract associated with quality of service characteristics [...]” [38].

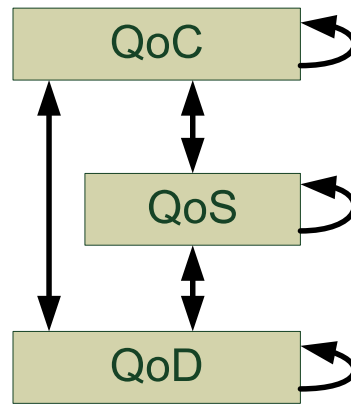


Figure 2.2: Interdependence between QoS, QoS, and QoD [17]

- Reconstructing the behaviour of context-aware systems: *“Context information is used to automatically adapt services or the content they provide. Therefore, the imperfection of context information has a significant impact on the experiences users make with CASs [context-aware system].”* [17].
- Selection of appropriate context providers: *“In many cases, a particular context information can be generated using different value chains. These value chains can be realized by different, competing context providers, each operating an infrastructure of context sources and sub-services in order to derive that particular context information. In such a scenario, it is not unlikely that competing context providers deliver the same context information with different QoC. From the point of view of a CAS provider, QoC is then a valuable indicator to select an appropriate context provider.”* [17].
- Adaptation of context refinement: *“Like context is used to modify the behavior of a CAS, QoC can be utilized to adapt the context refinement process. [...] an important task of context refinement is the derivation of new, high-level context information from low-level context information. Often, such a derivation is needed due to the unavailability of appropriate context sources. The quality of the respective low-level context information is an important indicator of whether or not the generation of high-level context information makes sense at all, and, if so, how to determine the quality of the produced context information.”* [17].
- Adaptation of context dissemination: *“Context dissemination comprises the distribution and storage of context information. Based on QoC agreements, a context provider can optimize the sub-processes of context dissemination.”* [17].
- Fine-grained privacy policies: *“[...] a context owner [...] represents the entity the context information is about. [...] a context owner can restrict access to his personal context. Without QoC the context owner could only determine who is allowed to access which part of his context. QoC, however, enables him to specify access policies in a much more fine-grained way. For example, a context owner might grant permission that a certain group might access his current location, but only with a precision of 10 kilometers and with a delay of some hours.”* [17].

Krause et al. define *“Quality of Context (QoC) is any inherent information that describes context information and can be used to determine the worth of the information for a specific*

*application. This includes information about the provisioning process the information has undergone ('history', 'age'), but not estimations about future provisioning steps it might run through."* [75].

In difference to the definition by Buchholz et al. [17], Krause et al. distinguish between the objective 'quality' and the application-specific 'worth' of context information and they use QoC to estimate the worth of context information for an application.

Additionally to the motivation for the usage of QoC, Krause et al. introduce challenges for modelling QoC:

- **Expressiveness of QoC parameters:** Similar to context information, QoC parameters can be expressed in several different representations. According to Krause et al., accuracy can be understood in several ways, namely as average error, minimal error, maximum error, or as probability distribution. Additionally, QoC parameters might also be missing or be incomplete.
- **Modelling QoC for context discovery:** Krause et al. motivate to avoid the hard-linking of context providers and context consumers and instead propose the usage of a dynamic context discovery at runtime. For that reason, descriptions of context provider have to be matched with the request for context information by context consumers in the context discovery phase. The context provider description should be used to decide whether a context provider can provide appropriate context information for a certain request. Context consumers can request context information either synchronously ('pull') or asynchronously ('push'). This means that the context provider immediately delivers the requested information respectively delivers only under certain circumstances. The problem here is, that if a context consumer sends a subscription for certain context information, it might express additional dynamic QoC dependencies in advance.

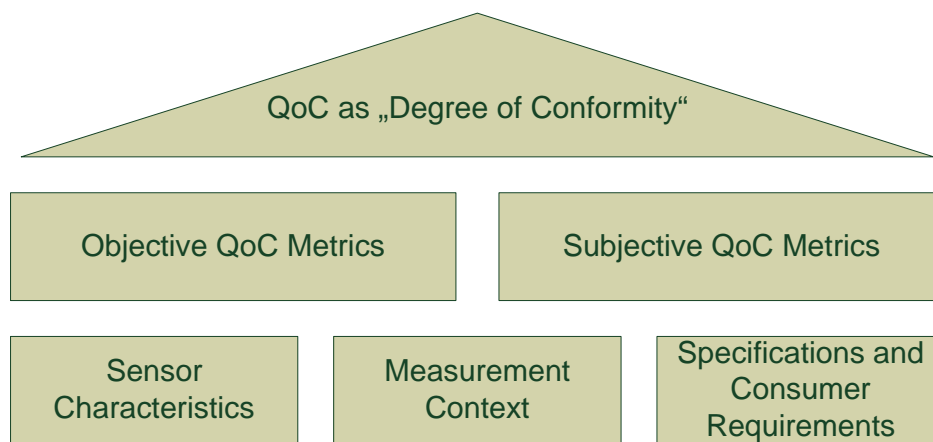
Sheikh et al. further motivate the usage of QoC in context-aware applications [123]. Based on the definition of Buchholz et al., they discussed five QoC indicators: precision, freshness, spatial resolution, temporal resolution and probability of correctness. According to Sheikh et al. there are three main reasons to consider QoC:

- Due to the imperfection of context information, Sheikh et al. additionally propose an adaptation of the behaviour of real-world context-aware applications to QoC information.
- *"Middleware efficiency - In a typical situation, there are several context sources available that can produce a certain type of context [...]. By comparing the required QoC and with the QoC of the available context sources [...], the middleware can optimize the selection of which context source (not) to use."* [123],
- *"Users' privacy enforcement - There is a relationship between the privacy sensitiveness of context information and its QoC. The middleware must therefore provide users (context owners) with the means to limit the QoC information provided to different requesters."* [123].

Manzoor et al. characterize QoC as *"Quality of Context indicates the degree of conformity of the context collected by sensors to the prevailing situation in the environment and the requirements of a particular consumer"* [88]. The authors base their definition on the statements by Buchholz et al. and Krause et al., but they argue that these explanations

only consider QoC “[...] as an objective term that is independent of the situation of use of context information and consumer requirements for that context information.”. For that reason, they introduce, additionally to the objective QoC metrics like accuracy, subjective QoC metrics like usability and significance taking into account the context consumer’s requirements.

Manzoor et al. further introduce a processing model for QoC information. As depicted in Figure 2.3 the model consists of several layers. The higher layers are calculated based on the data of the lower layers. The lowest layer consists of sensor characteristics (e.g. accuracy and time periods of measurements), the measurement context (e.g. measurement time and location) and specifications and consumer requirements (e.g. validity time and required attributes). The sensor’s characteristics and the measurement context is used for the calculation of the objective QoC metrics, whereas the calculation of subjective QoC metrics additionally involves the specifications and consumer requirements.



**Figure 2.3:** QoC Processing Model [88]

As aforementioned, several authors introduce different QoC dimensions (or indicators). Krause et al. already realized that “[...] there is a wide range of possible representation formats to express a QoC parameter. Labels like ‘precision’, ‘granularity’ [or] ‘probability of correctness’ can only be coarse categories regarding this variety [...]” [75]. We agree with Krause et al., as we also detected that authors give partly different names for similar QoC dimensions (similarity based on the definition) and that their definitions differ significantly in the degree of precision. For example, Buchholz et al. [17] give only raw, informal descriptions of their QoC dimensions, whereas Manzoor et al. [86, 88] provide very detailed and formal definitions of each dimension. At this point, we will only give a short overview of the most popular QoC dimensions as presented by Buchholz et al. [17], Sheikh et al. [123], and Manzoor et al. [86, 88]. Buchholz et al. and Sheikh et al. exclusively introduce the QoC dimensions stated below, whereas Manzoor et al. distinguish between QoC metrics [88] (called QoC parameters [86]) and sensor characteristics and the measurement context (called QoC sources [86]). QoC sources are values describing the context provider and its characteristics, like source location, life time, and measurement unit. The overview is depicted in Table 2.1.

Parameter	Authors	Description
Precision	Buchholz et al. [17]	<i>“precision describes how exactly the provided context information mirrors the reality.”</i>
	Sheikh et al. [123]	<i>“granularity with which the context information describes a real world situation.”</i>
Probability of correctness	Buchholz et al. [17]	<i>“[...] the probability that a piece of context information is correct.”</i>
	Sheikh et al. [123]	<i>“the probability that an instance of context accurately represents the corresponding real world situation, as assessed by the context source, at the time it was determined.”</i>
Trust-worthiness	Buchholz et al. [17]	<i>“[...] trust-worthiness also describes how likely it is that the provided information is correct. In comparison to correctness, however, trustworthiness is used by the context provider to rate the quality of the actor from which the context provider originally received the context information.”</i>
	Manzoor et al. [86]	<i>“[...] the belief that we have in the correctness of information in a context object. Trust-worthiness of a context object is highly affected by the space resolution, i.e., the distance between the sensor and the entity.”</i>
Resolution	Buchholz et al. [17]	<i>“[...] resolution denotes the granularity of information.”</i>
Up-to-datedness or freshness	Buchholz et al. [17]	<i>“[...] up-to-dateness describes the age of context information.”</i>
	Sheikh et al. [123]	Freshness is <i>“[...] the time that elapses between the determination of context information and its delivery to a requester.”</i> Based on the definitions of Buchholz et al. and Sheikh et al., up-to-datedness and freshness seems to be equivalent
	Manzoor et al. [86]	<i>“the degree of rationalism to use a context object for a specific application at a given time. We take into account the age of context object and the lifetime of that context object [...]”</i>
Temporal resolution	Sheikh et al. [123]	<i>“[...] the period to which a single instance of context information is applicable”</i>

Parameter	Authors	Description
Spatial resolution	Sheikh et al. [123]	<i>“[...] the precision with which the physical area, to which an instance of context information is applicable, is expressed”</i>
Completeness	Manzoor et al. [86]	<i>“[...] the completeness of a context object as the ratio of the sum of the weights of the available attributes of a context object to the sum of the weights of all the attributes of that context object”</i>

**Table 2.1:** Quality of Context Parameters

As a quintessence of the previous discussion about different QoC parameters, we can summarize that it is difficult to define a set of QoC parameters that fulfils all different opinions and requirements. Nevertheless, we do not focus on providing such a definition in this work. As QoC is one of the criteria for the selection and activation of a context provider, we need a rather generic definition of QoC. Based on the aforementioned definitions of Quality of Context, the following definition will be used in the context of this thesis:

**Definition 2.3** (Quality of Context). Quality of Context (QoC) is any information that describes the quality of information that is used as context information.

#### 2.2.4 Cost of Context

Quality of Context describes certainly only the part of the metadata regarding context information. Another important aspect is the cost for acquiring the context information. This does not necessarily need to be monetary but can also be interpreted as e.g. power consumption of the sensors for acquiring the information.

Villalonga et al. introduced the additional concept of Cost of Context (CoC), which they define as *“Cost of Context (CoC) is a parameter associated to the context that indicates the resource consumption used to measure or calculate the piece of context information.”* [137].

For this work we adopt the definition of Villalonga et al. and use the following definition:

**Definition 2.4** (Cost of Context). Cost of Context (CoC) is a parameter associated to the context that indicates the resource consumption used to measure or calculate the piece of context information.

Similar to QoC, we do not propose a concrete set of QoC parameters like CPU usage, memory footprint, monetary cost, . . .) but rather focus on providing support for arbitrary QoC and CoC parameters.

## 2.3 Self-Adaptation

As motivated in the definition of Ubiquitous Computing (Section 2.1), context awareness is solitary not sufficient to replace the reduced user interaction and to satisfy the vision of disappearing technology. The system has to react somehow to the changing conditions and thus to the retrieved context information. Therefore, it has several options like changing parameters, adding/exchanging/removing components or binding external services. This aspect is called self-adaptation. As stated by Henricksen et al., “*adaptation is required in order to overcome the intrinsically dynamic nature of pervasive computing*” [52]. After Geihs, adaptation itself can be divided into design time adaptation and runtime adaptation [41]. Whereas design time adaptation is static, adaptation at runtime is dynamic. The adaptation at runtime without any user interaction is called self-adaptation. Nzekwa et al. explain this as follows: “*Self-adaptive systems (or autonomic systems) are self-managing system that use Feedback Control Loops (FCLs) to monitor, analyse, plan, and act according to changes occurring in their environment.*” [99]. As stated by IBM, “*the function of any autonomic capability is a control loop that collects details from the system and acts accordingly.*” [61].

One of the most famous and earliest definition of self-adaptation, is the definition mentioned in an DARPA<sup>5</sup> announcement. According to the DARPA Broad Agency Announcement on Self-Adaptive Software, “*self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible*” [78].

Another definition is the one by Oreizy et al.: “*Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.*” [101].

There are many definitions for self-adaptation, but all refer more or less directly (and partially only implicitly) to context awareness. As we think that this is a key point, we adapt (and revise) the definition by Oreizy et al. The following definition will be used in the context of this thesis:

**Definition 2.5** (Self-Adaptation). Self-adaptive software modifies its own behaviour in response to changes in its context.

### 2.3.1 Adaptation Mechanisms

Software can be adapted in several ways. Geihs mentions parametrization and compositional adaptation as the most important ones [41]. McKinley et al. generalize the distinction to weak and strong adaptation: “*Weak adaptation involves modifying parameters (parameter adaptation) or performing low-cost/limited-impact actions, whereas strong adaptation deals with high-cost/extensive-impact actions, such as replacing components with those that improve system quality*” [92]. In the following, we focus

---

<sup>5</sup>DARPA: Defense Advanced Research Projects Agency, <http://www.darpa.mil>. Last visited Apr. 20, 2012.



on parametrization and compositional adaptation as the most prominent adaptation mechanisms. For more details see e.g. Salehie et al. [118].

Parametrization or parameter adaptation involves the modification of variables that determine program behaviour. As stated by Geihs, parameter adaptation does not modify the structure of the application but rather variables [41]. According to Khan, “*Parameter settings are defined at design time based on some ranges of values. For practical applications there are some constraints on choosing such value ranges. For example, a continuous value range would effectively create infinite number of parameter settings, eventually making it impossible to evaluate the appropriateness of particular setting. Choosing concrete values for parameter settings solves that issue.*” [65].

As mentioned by Khan, “[...] *compositional adaptation refers to the exchange of algorithmic or structural parts of a system in the aim of fitting it to the current environment*” [65]. This adaptation mechanism is particularly useful to adjust component-based applications. These applications are considered as a composition of components and alternative component implementations are used to realize particular functionalities of the application. Compositional adaptation cannot only influence the selection of specific component implementations but also the composition of components itself, so Geihs et al. [42]. Hence, an implementation of a component has not necessarily to be local. It can rather be realized by an external service, e.g. a web service as demonstrated for instance by Geihs et al. [154].

### 2.3.2 Adaptation Policies

The adaptation reasoning is part of the control loop (see e.g. [61]) and decides when an adaptation has to be triggered and also which “application mode” has to be activated. In general, three different types of adaptation policies exist: *action*, *goal* and *utility function policies* [63].

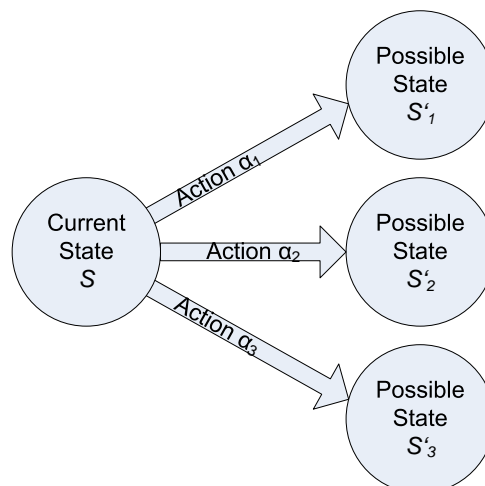


Figure 2.4: States and Actions

Kephart et al. introduce a framework based on the notions of states and actions in order to relate the different policy mechanisms [63]. A system is characterized as being in a

state  $S$  at a given moment in time, whereas  $S$  is a vector of attributes either measured by a sensor or inferred from low-level measurements. A policy will cause a transition/action  $\alpha$  from  $S$  to a new state  $S'$ .

Figure 2.4 depicts a scenario in which three transitions from state  $S$  exist, which means that the application has three different options to adapt its behaviour. Based on this example, the three different adaptation policies can be defined and distinguished as follows:

- **Action policies:** Based on the current state  $S$ , an action policy defines which action/transition  $\alpha$  has to be followed without taking into account the new state  $S'$ . This state will be reached by taking this action.
- **Goal policies:** In contrast to action policies, goal policies specify either a single desired state  $S'$  or one or more criteria that characterize an entire set of desired states. This also implies that any member of this set is equally acceptable as goal policies cannot express fine distinctions in preference.
- **Utility function policies:** Utility function policies can be seen as an extension of goal policies that map each possible state of a system to a real scalar value instead of the binary classification into desirable and undesirable states.

### 3 Logic and Ontologies

---

*“Contrariwise,” continued Tweedledee, “if it was so, it might be, and if it were so, it would be; but as it isn’t, it ain’t. That’s logic!”*

– Charles Lutwidge Dodgson (Lewis Carroll) (1832-1898)  
Through the Looking-Glass, and What Alice Found There (1871)

*“Logic is a fundamental organizing principle in nearly all areas in Computer Science. It runs a multifaceted gamut from the foundational to the applied. At one extreme, it underlies computability and complexity theory and the formal semantics of programming languages. At the other, it drives billions of gates every day in the digital circuits of processors of all kinds. Logic is in itself a powerful programming paradigm but it is also the quintessential specification language for anything ranging from real-time critical systems to networked infrastructures. It is logical techniques that link implementation and specification through formal methods such as automated theorem proving and model checking. Logic is also the stuff of knowledge representation and artificial intelligence” [24].*

Several types of logic exist, like classical, modal, and description logic. Propositional logic is the simplest form of (classical) logic consisting of simple propositions and conjunctions:

- $A = \text{Cats hate dogs}$
- $B = \text{Cats hate mice}$

These propositions can be either *true* or *false* (The proposition  $A$  is *true* where  $B$  is *false*.) First-order logic is a subclass of predicate logic which extends the propositional logic by quantifiers, predicate, and functional symbols. They allow to express e.g. that two objects are in relation to each other. For a detailed definition of syntax and semantics of both logics, we kindly refer to standard literature like Schönig [122].

In this chapter, we only focus on these parts of logics which are relevant for this thesis:

- Proof methods for propositional (see Section 3.1) and predicate logic (see Section 3.2) are used for the within the matching process of context offers and queries (see Chapter 8).
- Ontologies, in particular the Web Ontology Language (OWL) (see Section 3.3), serve as the basis of our context model (see Chapter 6) and for our context query and offer language, which reference to individuals and classes in the ontology (see Chapter 7).

## 3.1 Proof Methods for Propositional Logic

In this section we introduce methods to prove satisfiability of formulae in propositional logic. We can only give a short overview of a few methods. A detailed discussion would be out of scope and therefore we refer for more details to literature on logic like Schönig [122] or Smullyan [128]. We also skip the introduction of proving formulae with the truth table method. This simple but very costly method is explained in any general literature on logic like Schönig [122].

### 3.1.1 Analytic Tableaux for Propositional Logic

The analytic tableaux (or semantic tableaux) method is a decision procedure (a method for solving a decision problem) for propositional logic and a proof procedure (a systematic method for producing proofs) for formulae of first-order logic. The tableaux method can also determine the satisfiability of finite sets of formulae of various logics.

In this section we present the analytic tableaux method as introduced by Smullyan [128]. Smullyan based his method on the semantic tableaux method by Beth [12].

Before defining the analytic tableaux method, Smullyan introduces the following facts as the basis of his method (for any formulae  $X$  and  $Y$ )<sup>1</sup>:

1. a) If  $\neg X$  is true, then  $X$  is false.  
b) If  $\neg X$  is false, then  $X$  is true.
2. a) If a conjunction  $X \wedge Y$  is true, then  $X$  and  $Y$  are both true.  
b) If a conjunction  $X \wedge Y$  is false, then either  $X$  is false or  $Y$  is false.
3. a) If a disjunction  $X \vee Y$  is true, then either  $X$  is true or  $Y$  is true.  
b) If a disjunction  $X \vee Y$  is false, then  $X$  and  $Y$  are both false.
4. a) If an implication  $X \rightarrow Y$  is true, then either  $X$  is false or  $Y$  is true.  
b) If an implication  $X \rightarrow Y$  is false, then  $X$  is true and  $Y$  is false.

Based on the given facts, which directly follow the above definitions, Smullyan introduced a set of rules for the construction of the tableaux in the following schematic form:

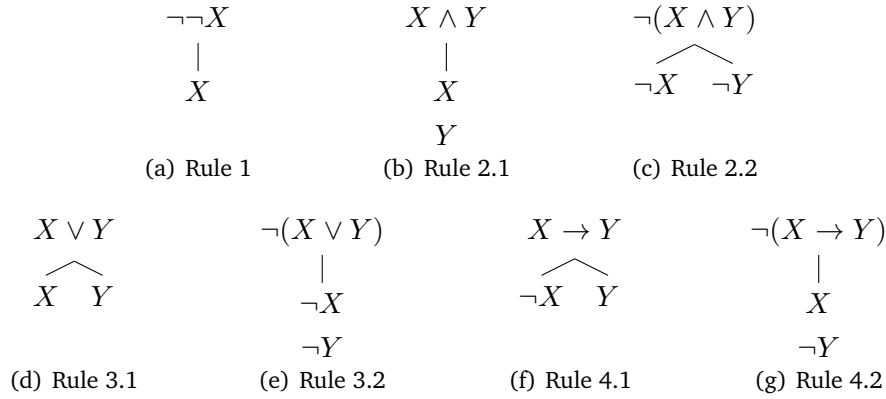
1. 
$$\frac{\neg\neg X}{X}$$
2. 
$$\frac{X \wedge Y}{\begin{array}{c} X \\ Y \end{array}} \quad \frac{\neg(X \wedge Y)}{\neg X \mid \neg Y}$$
3. 
$$\frac{X \vee Y}{X \mid Y} \quad \frac{\neg(X \vee Y)}{\begin{array}{c} \neg X \\ \neg Y \end{array}}$$

---

<sup>1</sup>In this work, we reuse the motivation and definitions of Smullyan [128] but transfer them into the notation of Schönig [122] which are also already used in the definitions above.

$$4. \frac{X \rightarrow Y}{\neg X \mid Y} \quad \frac{\neg(X \rightarrow Y)}{X \quad \neg Y}$$

Rule 1 means that from  $\neg\neg X$  we can directly infer  $X$ . Rule 2 indicates that  $X \wedge Y$  directly yields both  $X$  and  $Y$  whereas  $\neg(X \wedge Y)$  branches into  $X$  and  $Y$ . Rule 3 and 4 behave analogously. The visualization of these rules is shown in Figure 3.1



**Figure 3.1:** Analytic Tableaux for Different Construction Rules.

Smullyan distinguishes two types of formulae, which he calls ( $\alpha$ ) and ( $\beta$ ): type ( $\alpha$ ) are those that have *direct* consequences (viz.  $\neg\neg X$ ,  $X \wedge Y$ ,  $\neg(X \vee Y)$ ,  $\neg(X \rightarrow Y)$ ) while type ( $\beta$ ) *branches* (viz.  $\neg(X \wedge Y)$ ,  $X \vee Y$ ,  $X \rightarrow Y$ ). Furthermore he wrote: “It is practically desirable in constructing a tableau, that when a line of type ( $\alpha$ ) appears on the tableau, we simultaneously subjoin its consequences to all branches which pass through that line. Then that line need never be used again. And in using a line of type ( $\beta$ ), we divide all branches which pass through that line into sub-branches, and the line need never be used again” [128].

Smullyan also provides a precise definition of tableaux based on the previously mentioned construction rules [128]. This definition is based on trees. For that reason we have to add definitions of trees (more precisely ordered dyadic trees) before the definition of tableaux. The following definitions are also adopted from Smullyan.

**Definition 3.1** (Unordered Tree). An unordered tree  $\mathcal{T}$  is defined by the following items:

- A set  $S$  of elements called *points*.
- A function  $l$ , which assigns to each point  $x$  a positive integer  $l(x)$  called the *level* of  $x$ .
- A relation  $xRy$  defined in  $S$ , which is read “ $x$  is a *predecessor* of  $y$ ” or “ $y$  is a *successor* of  $x$ ”. This relation must obey the following conditions:

- $C_1$ : there is a unique point  $a_1$  of level 1. We call this point the *origin* of the tree.
- $C_2$ : every point other than the origin has a unique predecessor.
- $C_3$ : for any points  $x, y$ , if  $y$  is a successor of  $x$ , then  $l(y) = l(x) + 1$ .

We call a point  $x$  an *end point* if it has no successors; a *simple point* if it has exactly one successor, and a *junction point* if it has more than one successor. A *path* is defined as any finite or denumerable sequence of points, beginning with the origin. In this sequence of points, each term of the sequence (except the last, if there is one) is the predecessor of the next. A *maximal path* or *branch* describes a path whose last term is an end point of the tree or a path which is infinite. It follows from  $C_1, C_2, C_3$  that for any point  $x$  exists a unique path  $P_x$  whose last term is  $x$ . If  $y$  is within  $P_x$ , then we say that  $y$  *dominates*  $x$  or  $x$  is *dominated* by  $y$ . If  $x$  dominates  $y$  and  $x \neq y$ , then we say that  $x$  is *above*  $y$ , or that  $y$  is *below*  $x$ . We say that  $y$  is *between*  $x$  and  $z$  if  $y$  is above  $x$  or  $z$  and below the other.

**Definition 3.2** (Ordered Tree, Dyadic Tree). An *ordered tree*  $\mathcal{T}$  is defined as an unordered tree together with a function  $\theta$  which assigns to each junction point  $z$  a sequence  $\theta(z)$  which contains no repetitions and whose set of terms consists of all successors of  $z$ . Hence, if  $z$  is a junction point of an ordered tree, we can speak of the  $1^{st}, 2^{nd}, \dots, n^{th}, \dots$  successor of  $z$  (for any  $n$  up to the number of successors of  $z$ ) meaning the  $1^{st}, 2^{nd}, \dots, n^{th}, \dots$  terms of  $\theta(z)$ . For a simple point  $x$ , we also speak of the successor of  $x$  as the *sole* successor of  $x$ . An ordered tree in which each junction point has exactly two successors is called *dyadic tree*. For such trees we refer to the first successor of a junction point as the *left successor* and the second successor as the *right successor*.

**Definition 3.3** (Analytic Tableaux). An *analytic tableau* for  $X$  is an ordered dyadic tree, whose points are (occurrences of) formulae, which is constructed as follows: we start by placing  $X$  in the origin. Now suppose  $\mathcal{T}$  is a tableau for  $X$ , which has already been constructed; let  $Y$  be an end point. Then we may extend  $\mathcal{T}$  by either of the following two operations:

- (A) If some  $\alpha$ , as depicted in the first column of Table 3.1, occurs on the path  $P_Y$ , then we may adjoin  $\alpha_1$  or  $\alpha_2$  as the sole successor of  $Y$ .

$\alpha$	$\alpha_1$	$\alpha_2$
$X \wedge Y$	$X$	$Y$
$\neg(X \vee Y)$	$\neg X$	$\neg Y$
$\neg(X \rightarrow Y)$	$X$	$\neg Y$
$\neg\neg X$	$X$	$X$

**Table 3.1:** Operation (A) for Construction of an Analytic Tableau

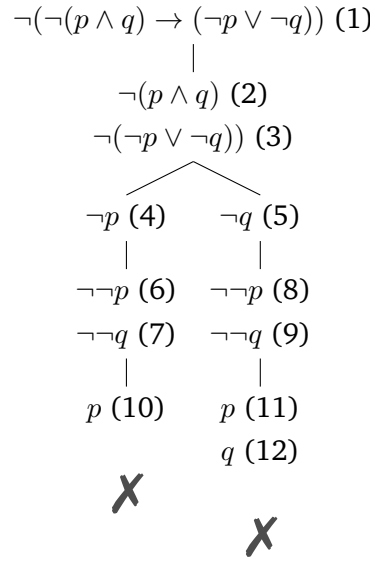
- (B) If some  $\beta$ , as depicted in the first column of Table 3.2, occurs on the path  $P_Y$ , then we may simultaneously adjoin  $\beta_1$  as the left successor of  $Y$  and  $\beta_2$  as the right successor of  $Y$ .

$\beta$	$\beta_1$	$\beta_2$
$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$X \vee Y$	$X$	$Y$
$X \rightarrow Y$	$\neg X$	$Y$

**Table 3.2:** Operation (B) for Construction of an Analytic Tableau

The above inductive definition of tableaux for  $X$  can be made explicitly as follows: given two ordered dyadic trees  $\mathcal{T}_1$  and  $\mathcal{T}_2$ , whose points are occurrences of formulae, we call  $\mathcal{T}_2$  a *direct extension* of  $\mathcal{T}_1$  if  $\mathcal{T}_2$  can be obtained from  $\mathcal{T}_1$  by one application of the operation (A) or (B) above. Then  $\mathcal{T}$  is a tableau for  $X$  iff there exists a finite sequence  $(\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n = \mathcal{T})$  such that  $\mathcal{T}_1$  is a 1-point tree whose origin is  $X$  and such that for each  $i < n$ ,  $\mathcal{T}_{i+1}$  is a direct extension of  $\mathcal{T}_i$ . A branch  $\theta$  of a tableau is *closed* if it contains some formula and its negation. And  $\mathcal{T}$  is called closed if every branch of  $\mathcal{T}$  is closed. A *proof* of a formula  $X$  means that the tableau for  $\neg X$  is closed.

**Example** In the following example we proof the tautology  $\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$ . The resulting tableau is depicted in Figure 3.2. The cross **X** under a branch indicates that this branch is closed as it contains a formulae and its contradiction.



**Figure 3.2:** Analytic Tableau for  $\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$

The tableau depicted in Figure 3.2 was constructed as follows. As defined in Definition 3.3, in order to proof a formula  $X$ , we have to check if the tableau for  $\neg X$  is closed. For that reason, the origin (marked with (1) in Figure 3.2) of our tableau for proving  $\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$  is  $\neg(\neg(p \wedge q) \rightarrow (\neg p \vee \neg q))$ . Now following the third  $\alpha$  rule  $\frac{\neg(X \rightarrow Y)}{X}$  we have to adjoin  $X$  and  $\neg Y$  to the tableau, which means we adjoin  $\neg(p \wedge q)$  (line (2)) and  $\neg(\neg p \vee \neg q)$  (line(3)). For the node in line (2), we now add node (4) and (5) following the first  $\beta$  rule  $\frac{\neg(X \wedge Y)}{\neg X \mid \neg Y}$ , which means the node in line (2) branches into  $\neg p$  and  $\neg q$ . Now the node in line (3) is further processed following the second  $\alpha$  rule  $\frac{\neg(X \vee Y)}{\neg X}$ . As explained above, the result of node (3) has to be added to all branches. Hence we adjoin  $\neg X$  (in our case  $\neg\neg p$  and  $\neg Y$  (in our case  $\neg\neg q$  to both branches of node (2) (thus to line (4) and (5)). After applying the forth  $\alpha$  rule  $\frac{\neg\neg X}{X}$

on node (6) and respectively on line (8) and (9), we have to adjoin  $p$  in line (10) ( $p$  in line (11) and  $q$  in line (12)). After adding  $p$  to the left branch (line (10)), we can close this branch (indicated by  $\times$ ). The branch is closed as it contains a formula (here  $p$  in line (10)) and its negation (here  $\neg p$  in line (4)) (see Definition 3.3). Following this rule we can also close the other branch as it contains both  $q$  and  $\neg q$ . As now all branches of the tableau are closed, the tableau is closed according to the definition.

**Definition 3.4** (Complete Branch, Complete Tableau). A branch  $\theta$  of a tableau is *complete* if for every  $\alpha$  which occurs in  $\theta$ , both  $\alpha_1$  and  $\alpha_2$  occur in  $\theta$ , and for every  $\beta$  which occurs in  $\theta$  at least one of  $\beta_1$  and  $\beta_2$  occurs in  $\theta$ . A tableau  $\mathcal{T}$  is called *completed* if every branch of  $\mathcal{T}$  is either closed or complete.

**Theorem 3.1.** If  $\mathcal{T}$  is any completed open tableau (open in the sense that at least one branch is not closed), the origin of  $\mathcal{T}$  is satisfiable. (For proof see Smullyan [128].)

### 3.1.2 Resolution for Propositional Logic

Resolution is a rule of inference leading to a refutation theorem-proving technique. Resolution exists for both propositional logic and first-order logic (see Section 3.2.1). To perform resolution it is necessary to have the formula in conjunctive normal form. For that reason, we adapt the definitions for normal forms by Schönig [122].

**Definition 3.5** (Normal Forms). A *literal* is an atomic formula or the negation of an atomic formula. A formula  $F$  is in conjunctive normal form (CNF) iff it is a conjunction of disjunctions of literals:

$$F = \left( \bigwedge_{i=1}^n \left( \bigvee_{j=1}^{m_i} L_{i,j} \right) \right) \text{ with } L_{i,j} \in \{A_1, A_2, \dots\} \cup \{\neg A_1, \neg A_2, \dots\}$$

A formula  $F$  is in disjunctive normal form (DNF) iff it is a disjunction of conjunctions of literals:

$$F = \left( \bigvee_{i=1}^n \left( \bigwedge_{j=1}^{m_i} L_{i,j} \right) \right) \text{ with } L_{i,j} \in \{A_1, A_2, \dots\} \cup \{\neg A_1, \neg A_2, \dots\}$$

**Theorem 3.2.** For every formula  $F$  in the propositional calculus, an equivalent formula in CNF and an equivalent formula in DNF exists. (For proof see Schönig [122].)

**Definition 3.6** (Clausal Form). A *clause* is a set of literals which is considered to be an implicit disjunction. A *unit clause* is a clause consisting of exactly one literal. A formula in *clausal form* is a set of clauses which is considered to be an implicit conjunction.

Hence, a formula  $F = (L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{k,n_k})$  in CNF with  $L_{i,j} \in \{A_1, A_2, \dots\} \cup \{\neg A_1, \neg A_2, \dots\}$  is equivalent to the following formula  $F' = \{\{L_{1,1}, \dots, L_{1,n_1}\}, \dots, \{L_{k,1}, \dots, L_{k,n_k}\}\}$  in clausal form.

**Definition 3.7** (Resolvent). Let  $K_1$ ,  $K_2$ , and  $R$  be clauses. Then  $R$  is called *resolvent* of  $K_1$  and  $K_2$  if a literal  $L$  exists with  $L \in K_1$  and  $\bar{L} \in K_2$  and  $R$  has the form:

$$R = (K_1 - \{L\}) \wedge (K_2 - \{\bar{L}\}).$$



$\bar{L}$  is defined as following:

$$\bar{L} = \begin{cases} \neg A_i, & \text{if } L = A_i, \\ A_i, & \text{if } L = \neg A_i. \end{cases}$$

**Definition 3.8** (Resolution Function  $Res$ ). Let  $F$  be a set of clauses. Then we define the resolution function  $Res(F)$  as follows:

$$\begin{aligned} Res(F) &= F \cup \{R \mid R \text{ is resolvent of two clauses in } F\} \\ Res^0(F) &= F \\ Res^{n+1}(F) &= Res(Res^n(F)) \text{ for } n \geq 0 \\ Res^*(F) &= \bigcup_{n \geq 0} Res^n(F). \end{aligned}$$

**Theorem 3.3** (Resolution theorem). A set of clauses  $F$  is unsatisfiable, iff  $\square \in Res^*(F)$ . (For proof see Schönig [122].)

Based on the resolution theorem, the recursive Algorithm 3.1 can be applied to decide whether a formula in CNF is satisfiable or not.

---

**Algorithm 3.1** Resolution Algorithm (returns true if formula is satisfiable)

---

**Precondition:** formula  $F$  in CNF

```

1: Built a set of clauses (also denoted with  $F$ ) based on  $F$ 
2: repeat
3:    $G \leftarrow F$ 
4:    $F \leftarrow Res(F)$ 
5: until ( $\square \in F$ ) or ( $F = G$ )
6: if  $\square \in F$  then
7:   return false
8: else
9:   return true
10: end if

```

---

## 3.2 Proof Methods for First-Order Logic

Similar to the propositional logic, several proof methods exist to proof first-order logic on satisfiability respectively unsatisfiability. In the following we will only discuss the most common methods, resolution and tableaux method.

### 3.2.1 Resolution for First-Order Logic

Similar to the resolution for propositional logic (see Section 3.1.2), resolution for first-order logic requires a formula to be specified in a specific form (for propositional logic it has to be in conjunctive normal form (CNF)). For that reason, we first have to define the required form. The following definitions are based on the definitions formulated by Schönig [122].

**Definition 3.9** (Clean Formula). A formula  $F$  is clean if no variable occurs both free and bound in  $F$  and every occurrence of quantifiers binds different variables.

**Definition 3.10** (Prenex Normal Form). A formula is called prenex or being in prenex normal form, if it is built as follows:

$$Q_1y_1Q_2y_2 \dots Q_ny_nF,$$

whereby  $Q_i \in \{\exists, \forall\}$ ,  $n \geq 0$  and  $y_i$  are variables. Furthermore,  $F$  does not contain any quantor.

**Theorem 3.4.** For every formula  $F$  exists an equivalent clean formula  $G$  in prenex normal form. (For proof see Schönig [122].)

**Definition 3.11** (Skolem Form). For every clean formula  $F$  in prenex normal form, we define the result of Algorithm 3.2 as its *skolem form*.

---

**Algorithm 3.2** Skolem Algorithm

---

- 1: **while**  $F$  contains  $\exists$  quantor **do**
  - 2:      $F$  has form  $F = \forall y_1 \forall y_2 \dots \forall y_n \exists z G$  for a clean prenex formula  $G$  and  $n \geq 0$
  - 3:     Let  $f$  be a n-ary function symbol, which is not part of  $F$
  - 4:      $F \leftarrow \forall y_1 \forall y_2 \dots \forall y_n G[z/f(y_1, y_2, \dots, y_n)]$
  - 5: **end while**
- 

**Definition 3.12** (Herbrand Universe). The *Herbrand universe*  $D(F)$  of a closed formula  $F$  in skolem form is the set of all variable-free terms, which can be built from the parts of  $F$ . In case that  $F$  does not contain a constant, we initially choose an arbitrary constant (for instance  $a$ ), and built the variable-free terms. Formally,  $D(F)$  is inductively defined as follows:

1. All constants in  $F$  are elements of  $D(F)$ . If  $F$  does not contain any constant,  $a \in D(F)$  applies.
2. For every n-ary function symbol  $f$  contained in  $F$  and terms  $t_1, t_2, \dots, t_n \in D(F)$ , the term  $f(t_1, t_2, \dots, t_n) \in D(F)$  applies.

**Definition 3.13** (Herbrand Structure). Let  $F$  be a formula in skolem form. Every structure  $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$  is called *Herbrand structure* of  $F$  if the following is valid:

1.  $U_{\mathcal{A}} = D(F)$ ,
2. For every n-ary function symbol  $f$  in  $F$  and  $t_1, t_2, \dots, t_n \in D(F)$ ,  $f^{\mathcal{A}}(t_1, t_2, \dots, t_n) = f(t_1, t_2, \dots, t_n)$

We call a Herbrand structure of a formula  $F$  a *Herbrand model*, if it is a model for  $F$ .

**Theorem 3.5.** Let  $F$  be a formula in skolem form.  $F$  is satisfiable, iff  $F$  has a Herbrand model. (For proof see Schönig [122].)

**Definition 3.14** (Herbrand Expansion). Let  $F = \forall y_1 \forall y_2 \dots \forall y_n F^*$  be a formula in skolem form. The *Herbrand expansion*  $E(F)$  is defined as following:  
 $E(F) = \{F^* [y_1/t_1][y_2/t_2] \dots [y_n/t_n] \mid t_1, t_2, \dots, t_n \in D(F)\}$

**Theorem 3.6** (Gödel-Herbrand-Skolem). Every formula  $F$  in skolem form is satisfiable, iff the set of formulae  $E(F)$  is satisfiable. (For proof see Schöning [122].)

**Definition 3.15** (Gilmore's Algorithm). Let  $F$  be a closed formula in skolem form and let  $\{F_1, F_2, F_3, \dots\}$  be an enumeration of  $E(F)$ . Algorithm 3.3 has the property to stop in finite time for unsatisfiable formulae.

---

**Algorithm 3.3** Gilmore's Algorithm

---

```

1:  $n \leftarrow 0$ 
2: repeat
3:    $n \leftarrow n + 1$ 
4: until  $(F_1 \wedge F_2 \wedge \dots \wedge F_n)$  is unsatisfiable  $\triangleright$  This can be checked with any method for
   propositional logic
5: return 'unsatisfiable' and halt

```

---

**Definition 3.16** (Ground Clauses). A *ground term* is a term without occurrences of variables. A *ground formula* is a formula in which only ground terms occur. A *predicate clause* is a disjunction of atomic formulae. A *ground clause* is a disjunction of ground atomic formulae. A *ground instance* of a predicate clause  $K$  is the result of substituting ground terms for the variables of  $K$ . This substitutions are called *ground substitutions*.

**Definition 3.17** (Ground Resolution Algorithm). Let  $F$  be a closed formula in skolem form with matrix  $F^*$  in CNF and let  $\{F_1, F_2, F_3, \dots\}$  be an enumeration of  $E(F)$ . Algorithm 3.4 is called *Ground resolution algorithm*.

---

**Algorithm 3.4** Ground Resolution Algorithm

---

```

1:  $i \leftarrow 0$ 
2:  $M \leftarrow \emptyset$ 
3: repeat
4:    $i \leftarrow i + 1$ 
5:    $M \leftarrow M \vee \{F_i\}$ 
6:    $M \leftarrow Res^*(M)$ 
7: until  $\square \in M$ 
8: return 'unsatisfiable' and halt

```

---

**Theorem 3.7** (Ground resolution theorem). A formula  $F = \forall y_1 \dots \forall y_n F^*$  with matrix  $F^*$  in CNF is unsatisfiable iff there is a set of ground clauses  $K_1, \dots, K_m$  such that:

- $K_m$  is the empty clause, and
- for every  $i = 1, \dots, m$ 
  - either  $K_i$  is a ground instance of a clause  $K \in F^*$ , which means  $K_i = K[y_1/t_1] \dots [y_n/t_n]$  with  $t_1, \dots, t_n \in D(F)$ ,
  - or  $K_i$  is a resolvent of two clauses  $K_a, K_b$  with  $a < i$  and  $b < i$ .

(For proof see Schöning [122].)

### 3.2.2 Analytic Tableaux for First-Order Logic

The analytic tableaux method for first-order logic presented here, is based on the method of Smullyan and is an extension of the analytic tableaux method for propositional logic as presented in Section 3.1.1 [128]. Smullyan therefore uses the  $\alpha$  and  $\beta$  rules exactly as he did for propositional logic and additionally adds the two categories  $\gamma$  and  $\delta$ .  $\gamma$ -formulae (also formulae of universal type) are any formulae of the form  $(\forall x)A$  and  $\neg(\exists x)A$  and for any parameter  $a$ ,  $\gamma(a)$  means  $A[x, a]$ ,  $\neg A[x, a]$  respectively.  $\delta$ -formulae (also formulae of existential type) are any formulae of the form  $(\exists x)A$  and  $\neg(\forall x)A$ , and  $\delta(a)$  stands for  $A[x, a]$ ,  $\neg A[x, a]$  respectively. The tableaux rules are extended accordingly resulting in the following four rules:

$$\text{Rule A: } \frac{\alpha}{\alpha_1 \quad \alpha_2}$$

$$\text{Rule B: } \frac{\beta}{\beta_1 \mid \beta_2}$$

$$\text{Rule C: } \frac{\gamma}{\gamma(a)}, \text{ where } a \text{ is any parameter.}$$

$$\text{Rule D: } \frac{\delta}{\delta(a)}, \text{ where } a \text{ is a new parameter, or } a \text{ has not been previously introduced by rule D, and does not occur in } \delta, \text{ and no parameter of } \delta \text{ has been previously introduced by rule D.}$$

The rules A and B are the same as in propositional logic, whereas rule C and D can be concreted as following:

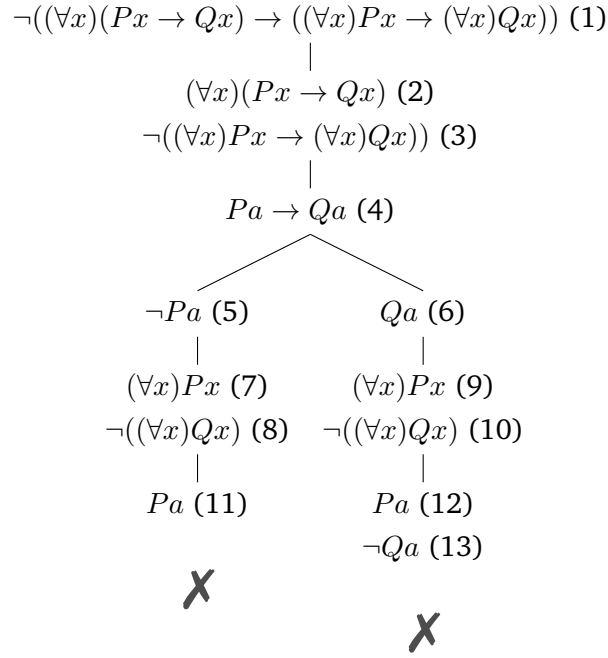
$$\text{Rule C: } \frac{(\forall x)A}{A[x/a]} \quad \frac{\neg(\exists x)A}{\neg A[x/a]}$$

$$\text{Rule D: } \frac{(\exists x)A}{A[x/a]} \quad \frac{\neg(\forall x)A}{\neg A[x/a]}$$

**Example** In the following example we proof the formula  $(\forall x)(Px \rightarrow Qx) \rightarrow ((\forall x)Px \rightarrow (\forall x)Qx)$ . The resulting tableau is depicted in Figure 3.3.

The tableau was constructed as follows. Following Definition 3.3, to proof a formula  $X$ , we have to check if the tableau for  $\neg X$  is closed. Hence, the origin depicted in line (1) of our tableau for proving the formula  $(\forall x)(Px \rightarrow Qx) \rightarrow ((\forall x)Px \rightarrow (\forall x)Qx)$  is the formula  $\neg((\forall x)(Px \rightarrow Qx) \rightarrow ((\forall x)Px \rightarrow (\forall x)Qx))$ . Now following the third  $\alpha$  rule  $\neg(X \rightarrow Y)$

$\frac{X}{\neg Y}$  we have to adjoin  $X$  and  $\neg Y$  to the tableau, hence we add  $(\forall x)(Px \rightarrow Qx)$  (line(2)) and  $\neg((\forall x)Px \rightarrow (\forall x)Qx)$  (line(3)). For line (2), we now add line (4) following the first  $\gamma$  rule  $\frac{(\forall x)A}{A[x/a]}$ . For that reason, line (2) leads to  $Pa \rightarrow Qa$ .



**Figure 3.3:** Analytic Tableau for  $(\forall x)(Px \rightarrow Qx) \rightarrow ((\forall x)Px \rightarrow (\forall x)Qx)$

Following the third  $\beta$  rule  $\frac{X \rightarrow Y}{\neg X \mid Y}$ , line (4) branches into  $\neg Pa$  and  $Qa$ . To these two

branches the results by applying the third  $\alpha$  rule  $\frac{\neg(X \rightarrow Y)}{X}$  on line (3) are appended.

For that reason, we have to add  $(\forall x)Px$  (line (7) and (9)) and  $\neg((\forall x)Qx)$  (line (8) and (10)) to line (5) and (6) respectively. Line (7) results in  $Pa$  (line (11)) after using the

first  $\gamma$  rule  $\frac{(\forall x)A}{A[x/a]}$ . This closes the branch as it contains both  $Pa$  (line (11)) and  $\neg(Pa)$

(line (5)). The second branch can be closed analogously after applying the second  $\delta$  rule  $\frac{\neg(\forall x)A}{\neg A[x/a]}$  on line (10).

On the previous pages we have presented the analytic tableaux and the resolution methods for both propositional and first-order logic. These methods of giving evidence of the satisfiability of a formula are used within the matching process. A detailed description and discussion about which method is finally used and why it is used can be found in Section 8.3.

### 3.3 Ontologies

In this thesis, we focus on context awareness in ubiquitous computing environments. Context sources are distributed on several heterogeneous devices. These devices are equipped with a set of heterogeneous context sensors and other sources containing context information, e.g. a calendar or a contact database. One of the goals of this thesis is the seamless integration of such heterogeneous context sources and to support the

interoperability between context sources and consumers. Therefore, context services and consumers must have a common knowledge about the exchanged information. A common vocabulary prevents using the same term with different semantics. However, for the exchange of heterogeneous context information a common vocabulary is a good starting point. Also, information about the relationship between the different items is required to support context reasoning and finally, knowledge about the representation of the information is required. Otherwise it would be possible, that e.g. context consumer and provider have the same understanding of the term *position* but the consumer expects to receive a position in form of an address and the provider sends the position in WGS84 coordinates. This knowledge is described in an ontology. In this section we will give a first introduction and general overview on ontologies, whereas our concrete ontological concepts will be discussed in Chapter 6.

### 3.3.1 General Discussions on Ontologies

In the *Handbook on Ontologies* by Staab et al. [130], Guarino et al. provide a detailed introduction on ontologies: *“The word ‘ontology’ is used with different senses in different communities. The most radical difference is perhaps between the philosophical sense, which has of course a well-established tradition, and the computational sense, which emerged in the recent years in the knowledge engineering community, starting from an early informal definition of (computational) ontologies as ‘explicit specifications of conceptualizations’”* [45].

According to Gruber *“[...] a body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them [...]. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly. An ontology is an explicit specification of a conceptualization”* [44].

Ding et al. mention that conceptualization is only one of three important aspects that are needed to realize explicit ontologies [35]:

- *Conceptualization: “The language should choose an appropriate reference model, such as entity-relationship model and object-oriented model, and provide corresponding ontology constructs to represent factual knowledge, such as defining the entities and relations in a domain, and asserting relations among entities.”*
- *Vocabulary: “Besides the semantics, the language should also cover the syntax such as symbol assignment (i.e., assigning symbols to concepts) and grammars (i.e., serializing the conceptualism into explicit representation).”*
- *Axiomatization: “In order to capture the semantics for inference, rules and constraints are needed in addition to factual knowledge. For example, we can use rules to generate new facts from existing knowledge, and to validate the consistency of knowledge.”*

Additionally to these general requirements on an ontology language, Ding et al. introduce some aspects in order to consider *‘web-based knowledge sharing activities’*:

- *Extensibility: to support the incremental development of ontologies and to ease the reuse of existing ontological concepts, ontologies should be extensible.*

- *Visibility*: “Merely publishing knowledge on the Web does not guarantee that it can be readily understood by machines or human users. In order to make knowledge visible on the Web, additional common ontological ground on syntax and semantics is required between information publishers and consumers. This requirement is especially critical to machines since they are not capable of understanding knowledge written in an unfamiliar language” [35].
- *Inferenceability*: ontologies do not only contain existing knowledge, but also support the computation of new knowledge based on the existing knowledge, like “enabling logical inference on facts through axiomatization” [35].

Ding et al. argue that these additional requirements caused the evolution of ontologies especially in the *Semantic Web* community. As the above mentioned requirements also meet our needs regarding an ontology, we will focus on ontology languages common in the semantic web community.

Whereas traditional ontology languages like F-Logic<sup>2</sup> [66], KIF & Ontolingua<sup>3</sup> [44], LOOM [85], and OCML<sup>4</sup> [95] are mainly used to achieve interoperability of knowledge-based systems in artificial intelligence, the web-based ontology languages have been developed to make steps forward to the vision of a semantic web<sup>5</sup>. Hence, languages like DAML+OIL [28], Web Ontology Language (OWL) [139, 140], RDF<sup>6</sup> [141], and RDF-S<sup>7</sup> [142] have been mainly developed to facilitate the interchange of ontologies across the World Wide Web (WWW) and the cooperation among heterogeneous agents placed on it.

From this set of languages, OWL turns out to be the de-facto standard, which has also been confirmed by Antoniou et al. [7]. They summarize the development and evolution of OWL starting from RDF and DAML+OIL as follows: “*The expressivity of RDF and RDF Schema [. . .] is deliberately very limited: RDF is (roughly) limited to binary ground predicates, and RDF Schema is (again roughly) limited to a subclass hierarchy and a property hierarchy, with domain and range definitions of these properties. However, the Web Ontology Working Group of W3C identified a number of characteristic use-cases for Ontologies on the Web which would require much more expressiveness than RDF and RDF Schema. A number of research groups in both America and Europe had already identified the need for a more powerful ontology modelling language. This led to a joint initiative to define a richer language, called DAML+OIL [. . .]. DAML+OIL in turn was taken as the starting point for the W3C Web Ontology Working Group in defining OWL, the language that is aimed to be the standardised and broadly accepted ontology language of the Semantic Web*” [7].

---

<sup>2</sup>Frame Logic.

<sup>3</sup>Ontolingua is based on KIF.

<sup>4</sup>Operational Conceptual Modelling Language.

<sup>5</sup>According to the W3C, “The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF).” [138].

<sup>6</sup>Resource Description Framework

<sup>7</sup>RDF Schema

### 3.3.2 OWL

Figure 3.4 shows the so-called *Semantic Web Layer Cake* also known as *Semantic Web Stack* [15]. It shows the different languages and technologies that are used to ‘build’ the semantic web. Each layer exploits and uses capabilities of the layers below. As depicted, OWL builds on RDF-S, which is based on RDF, Extensible Markup Language (XML), and finally the Uniform Resource Identifiers (URI) respectively the Internationalized Resource Identifier (IRI) concepts. We will give a short overview of how the technologies underlying OWL are used, whereas we skip the discussions of other technologies and concepts, as they are out of scope.

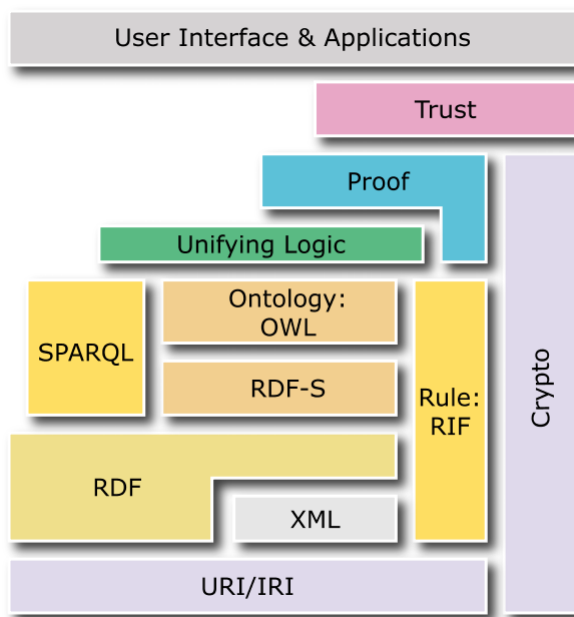


Figure 3.4: Semantic Web Layer Cake [15]

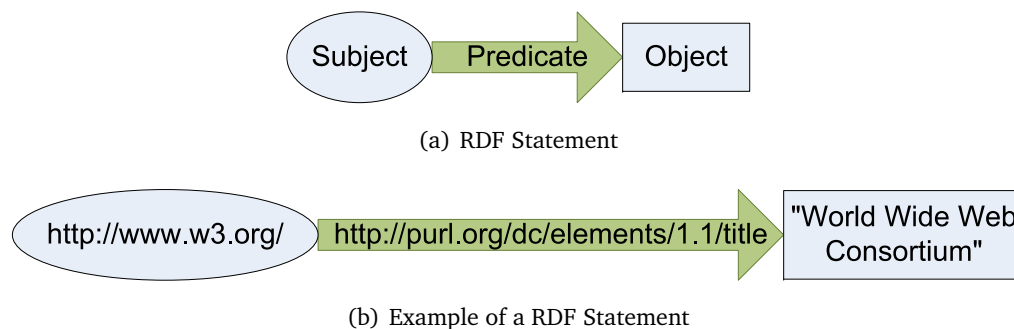
The bottom layer contains the URI/IRI concepts. The *Uniform Resource Identifiers (URI)* have basically two functions. They provide unique, unambiguous names for arbitrary concepts and they can be used as a reference to those concepts (or at least a description). The *Internationalized Resource Identifier (IRI)* is a generalized form of the URI, which allows all Unicode characters instead of only a subset of ASCII characters.

The *Resource Description Framework (RDF)* is a language used to represent information on resources in the web [141]. These resources are described in terms of *properties* and *property values* using *RDF statements*. A statement is represented as a triple consisting of *subject*, *predicate*, and *object* as shown in Figure 3.5. A concrete example<sup>8</sup> is depicted in the second subfigure of Figure 3.5 explaining that the title of the the website `http://www.w3.org/` is ‘World Wide Web Consortium’.

The *subject* in general describes a resource. As stated by Lacy, “[...] resources are the key building block in RDF. Resources are typically associated with nouns (i.e. people, places, things). An RDF is anything with an associated URIsref. Since anything can have a URIsref,

<sup>8</sup>The example is inspired from the W3C RDF Validation service hosted at `http://www.w3.org/RDF/Validator/`. Last visited on Aug 23, 2012.





**Figure 3.5:** RDF Statement and Example

anything can be a resource” [77]. The *predicate* of the RDF statement describes a *property*. “Properties describe attributes of resources and relationships between resources” [77]. So, properties link either resources with data values or they associate resources with other resources in order to describe their relationship and the *object* of the RDF statement is either a data value or a resource.

Even if RDF and RDF-S provide basic capabilities for describing resources, several capabilities are missing. For example, RDF and RDF-S do not support restrictions on property cardinality. Furthermore it is not possible to express disjoints between classes of resources. To overcome these shortcomings, OWL [139, 140] has been developed. It builds on RDF, more precisely OWL semantically extends RDF respectively RDF-S.

The W3C standard for OWL foresees three different variants of the language, each providing a different level of expressiveness. These are OWL Full, OWL DL and OWL Lite. In OWL Full, all language elements and constructs can be used as long as the result is valid RDF. Consequently, it also covers OWL DL and OWL Lite. In general, OWL Full is undecidable.

The undecidability of OWL Full was the main reason to standardize other variants of OWL. In difference to OWL Full, OWL DL and OWL Lite are decidable. In comparison to OWL Full, resources in OWL DL are only allowed to be either a class, a data type, a property of an individual, a data value or part of the built-in vocabulary [130]. Furthermore, all resources must be explicitly typed. Another restriction of OWL DL is that object and data type properties are disjoint and the cardinality restrictions for transitive properties are not allowed. Finally, the usage of anonymous classes is limited to the domain and range of owl:equivalentClass and owl:disjointClass and to the range of owl:subClassOf.

OWL Lite is a sub-language of OWL DL and also of OWL Full. Additionally to the restrictions of OWL DL, it is not allowed to use owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf and owl:hasValue. Furthermore, cardinality can only be limited to 0 or 1 and the usage of anonymous classes in both domain and range of owl:equivalentClass is not allowed.

According to Hitzler et al. OWL DL is the most used variant of OWL [55]. OWL Full is avoided due to its undecidability, whereas OWL Lite is avoided due to its limited expressiveness.



## 4 Related Work

---

*The young always have the same problem - how to rebel and conform at the same time. They have now solved this by defying their parents and copying one another.*

– Quentin Crisp (1908-1999)  
The Naked Civil Servant (1968)

In this chapter ‘Related Work’, existing approaches for supporting the development of context-aware applications<sup>1</sup> are discussed and evaluated according to the requirements described in Section 1.2:

- Requirement 1: Semantic discovery and integration of independently developed context services and consumers
- Requirement 2: Loose coupling of context providers and consumers
- Requirement 3: Exchange and interpretation of heterogeneously represented context information
- Requirement 4: Expressing context offers and needs
- Requirement 5: Activation and deactivation of local context services
- Requirement 6: Dynamic selection of context services based on QoC and CoC
- Requirement 7: Minimizing total amount of resources used by context services

2007 Baldauf et al. published an extensive survey on context-aware systems [8]. This survey gives a very good overview of the different definitions of context awareness and common architecture principles for context-aware systems. In their survey, the authors presented an evaluation of different existing context-aware frameworks. For the evaluation, they introduced several different evaluation criteria:

- The **architecture** of a context-aware system is mainly driven by the context acquisition method. The main criteria for a reasonable architectural approach is the separation of concerns between the context acquisition and the components with user interaction as proposed by Abowd et al. [2]. This corresponds to our request for loose coupling (see Requirement 2).
- The **sensing** technology is implemented differently in every framework. It is important, that the concrete sensing mechanisms are encapsulated in separate components to support the aforementioned separation of concerns. Furthermore,

---

<sup>1</sup>We use the term *context-aware systems* as umbrella term for frameworks and middleware supporting the development of context-aware applications.

it allows to access the contextual data via defined interfaces. This is implicitly requested by our requirement for semantic discovery (see Requirement 1), the ability to express context offers and needs (see Requirement 4) and loose coupling (see Requirement 2), as it is only possible to describe, discover, select and (de-) activate specific context sources if these are separated into explicit components.

- Also the underlying **context model** has an important influence on the context-aware system. Baldauf et al. reference the survey of Strang et al., who evaluate the different kinds of context models [133]. This survey was also an important input when developing our context model. As the exchange of heterogeneously represented context information is an important aspect of our system (see Requirement 3), the criteria of the underlying context model is implicitly part of our requirement for supporting heterogeneity.
- The criteria of the underlying **context processing** logic refers to the abilities of interpreting, inferring, aggregating, or composing context information in order to retrieve higher-level information. This is also requested by the loose coupling as described in Requirement 2.
- **Resource discovery** mechanisms are currently used rarely in the different existing frameworks. Such dynamic mechanisms are important, especially in a pervasive environment, where the availability of sensors and other context sources changes rapidly (new ones are added and removed). This corresponds to our requirement on semantic discovery (see Requirement 1).
- Access to **historical context data** might be partially necessary to establish trends and to predict future context values. This is required to some extent for our requested support for activation and deactivation (see Requirement 5) as without knowledge about historical data (more precisely historical metadata) it is only possible to select deactivated sources on default/average values.
- As context may include sensitive information on e.g. people, it is necessary to have the opportunity to protect **privacy**. This criteria is not addressed in this thesis but rather part of our future work (see Section 13.2).

Based on these criteria, the Baldauf et al. evaluated the several frameworks, namely the Context Managing Framework [73], SOCAM [46], CoBrA [21, 19], the Context Toolkit [117, 33, 2], Hydrogen [56], the Sentient Object Model [13], and Gaia [110]. Most of the criteria from Baldauf et al. correspond to our requirements. However, some of them (like Requirement 6 on dynamic selection) are not considered by Baldauf et al. For that reason, we refresh the evaluation of the above mentioned works and add some additional relevant systems. In this chapter we only consider these approaches which focus on the development of context-aware systems. Approaches which provide solutions to only one specific requirement are not discussed in this chapter but in the different discussions at the end of the chapters of Part II.

## 4.1 Existing Context-Aware Systems

In this section we present and evaluate several existing context-aware systems based on the requirements introduced in Section 1.2. The systems are presented in alphabetic order.

### 4.1.1 Adaptive middleware for context-aware applications in smart-homes

Huebscher et al. present in their paper [59] a middleware for developing context-aware applications. This middleware exhibits autonomic properties and can choose one of several context providers offering the same type of context according to the requirements of the context consumer. The adaptive middleware uses utility functions to determine which alternative should be used at any time. For this dynamic selection, Huebscher et al. introduce context services<sup>2</sup>, which connect application and different context providers. These providers, connected by a context service offer the same type of context but use different underlying sensors. This concept of context services allows an application to use a type of context whilst abstracting from the actual instance of a context provider. A directory service is used by the context services to find relevant context providers. When context providers join the network, they advertise their presence to the directory service, supplying it with the descriptive attributes for the service they provide. The directory service monitors the status of the context providers and can send notification events to the adaptation engine to trigger a new selection of the context providers.

**Major contributions:** Huebscher et al. provide a utility-based approach for the selection of context sources. Their approach dynamically reacts on changes of the provided quality of the different context providers.

**Weaknesses:** Even if Huebscher et al. mention that “[...] context acquisition is a costly operation [...]”, they address only implicitly the issue of cost minimization by selecting only one context provider per context request. However, while selecting context services, cost related metadata are not considered. Furthermore, they do not address the issue of dynamic activation and deactivation of context sensors and the according challenges (e.g. calculation of the QoC of a deactivated context source). Finally the approach requires a common understanding of parameters and the dimensions these parameters are specified in.

### 4.1.2 ASC-CoOL & CoCo

Strang et al. developed a context modelling approach called Aspect Scale Context (ASC) model, which is based on ontologies [134]. Based on this model, the authors derived the Context Ontology Language (CoOL), which is used to enable context awareness and contextual interoperability during service discovery and execution in a distributed architecture. *“An aspect is a classification, symbol- or value-range, whose subsets are a subset of all reachable states, grouped in one or more related dimensions called scales. [...] A scale is an unordered set of objects defining the range of valid context information”* [134]. Hence, an aspect corresponds to what we call the information type or scope and the scale to what we call context representation. An example for an aspect is GeographicCoordinateAspect and two corresponding scales could be WGS84Scale or GaussKruegerScale. In this respect, the work of Strang et al. stands out from all other analysed approaches, as it also captures different representations for a type of context information. Besides, transformations between different representations are covered by

---

<sup>2</sup>The concept of context services is used here differently as in our work. Whereas Huebscher et al. use context services as layer to group similar context sources, in our solution, context services encapsulate arbitrary sources and provide a common interface and service description for these sources.

IntraOperations and InterOperations. IntraOperations just convert between two scales of one aspect, whereas InterOperations also involve information corresponding to other aspects. The CoCo (Context Composition) system by Buchholz et al. relies on the context model of Strang et al. [16]. In their work, context consumers can query for context information with a so-called CoCoGraph, which is a manually constructed chain for composing and transforming context information.

**Major contributions:** The ASC-CoOL approach by Strang et al. provides an ontology-based context model, which explicitly addresses heterogeneity with regard to different representations of context information. Furthermore, the system allows the transformations between different scales of an aspect.

**Weaknesses:** Even if explicitly mentioning metadata for context information, the authors do not explain how this metadata are handled when performing conversions between different scales. The conversions called IntraOperations and InterOperations are also explained very imprecisely. Other required aspects like context selection are not addressed by the approach (also not by the CoCo system). The issue of using QoC during the discovery and matching of useful context providers has been discussed by Krause et al. But according to them, it has not been implemented in the CoCo system [75].

### 4.1.3 AWARENESS

In the AWARENESS project<sup>3</sup>, Sinderen et al. and Sheikh et al. motivate the consideration of QoC and identify five relevant QoC indicators for context-aware middleware [125, 123, 124]. Thereby the authors reuse and extend the motivation and definition of QoC by Buchholz et al. [17]. Whereas Buchholz et al. describe six use cases for QoC, Sheikh et al. generalize and categorize these use cases to three:

- QoC-based application adaptation: the application adapts its behavior according to the QoC information of the incorporated context sources.
- Middleware efficiency: the middleware selects the “best” context source from a set of context sources offering the same type of context information based on the QoC.
- Users’ privacy enforcement: according to the privacy requirements of the users, the middleware limits the QoC of the delivered context information.

After introducing these three application areas for QoC, the authors discussed five QoC indicators and their quantification: precision, freshness, temporal resolution, spatial resolution, and probability of correctness. Moreover, they justify the selection of these indicators and why they did not take into account other indicators like trustworthiness which have already been identified as relevant by other researchers. In their work the authors mainly focus on the protection of the user’s privacy.

**Major contributions:** Sheikh et al. motivate clearly the usage of QoC and also they justify the selection of QoC indicators. In their work, the authors motivate the adaptation/reduction of the quality of context information to protect the user’s privacy.

**Weaknesses:** Even if they provide a clear motivation for the usage of QoC, they only focus on the protection of the user’s privacy but do not provide any solution how to

---

<sup>3</sup>AWARENESS (Context AWARE mobile NEtworks and ServiceS) <http://www.freeband.nl/project.cfm?language=en&id=494>

solve the raised issues of QoC-based application adaptation and middleware efficiency. Furthermore, by splitting the QoC indicator ‘resolution’ introduced by Buchholz et al. [17] into spatial and temporal resolution, an important subset of this indicator is lost. For example, it is not possible to express the resolution of a temperature sensor.

#### 4.1.4 CARE

The *Context Aggregation and REasoning (CARE)* middleware aims at supporting context-aware adaptation of Internet services in a mobile computing environment. As described by Agostini et al., the platform focuses on the server-side adaptation of services providing content [3, 5, 4]. The authors distinguish between shallow context data (e.g. device capabilities) and non-shallow context data (e.g. user activity). They are using a logic programming language for reasoning shallow context data, while they have adopted OWL DL as the language for representing and reasoning with ontology-based context data. The context data are managed by the different entities (e.g. user, network operator, service provider). A profile contains all context data collected and managed by a certain entity and includes both shallow context data and ontology-based context data which are expressed by means of references to ontological classes and relations. Both the user and the service provider can declare policies in the form of rules over context data in order to derive higher-level context data and to determine the adaptation parameters of the service. A dedicated module, the context provider, is in charge of building the aggregated context data for the application logic, evaluating adaptation policies and solving possible conflicts. While the initial version of the approach only presents the support for shallow context data based on Composite Capability/Preference Profiles (CC/PP) [68], the extended version [5] contains a hybrid approach. In this approach, ontology reasoning is loosely coupled with the efficient rule-based reasoning of a middleware architecture for service adaptation. While rule-based reasoning is performed at the time of a service request to evaluate adaptation policies and reconcile possibly conflicting context information, ontology reasoning is mostly performed asynchronously by local context providers to derive non-shallow context information. The context model is based on CC/PP profiles. The authors are using the CC/PP profile for shallow profile data (context data which can be modelled in a natural way by using attribute/value pairs, and where the semantics for attributes and their allowed values is clear) and an extended CC/PP profile linking those attributes modelling non-shallow context data (e.g. user activities) to ontology concepts that formally define their semantics.

**Major contributions:** The CARE middleware focuses on the server-side adaptation of services based on context information provided by the user and the communication provider. It contains a very efficient and scalable approach for combining rule-based reasoning with ontology-based reasoning. Ontology-based reasoning in general is known to be very resource consuming. As a consequence, the authors splitted the ontology in to several parts: *“The solution we adapt consists in keeping the terminological part of the ontology [...] static, in order to be able to perform the TBox classification in advance to the service request. [...] Furthermore, the assertional part of the ontology can be filled in advance to the service request with those instances that are known a priori [...]”* [4].

**Weaknesses:** The middleware is built to adapt content providers based on context information about the requesting user. For these adaptations, context information is collected from the user’s device, the communication provider, and the service provider.

During this process neither quality of context information nor cost of context information are used, for example to solve conflicts when the user's device provides the position retrieved by the GPS sensor and the communication provider provides a position retrieved by triangulation. Furthermore, the authors do not support different representations of context information.

#### 4.1.5 CoBrA

The Context Broker Architecture (CoBrA) by Chen et al. is an architecture to support context-aware systems in smart spaces [21, 22, 19, 20]. A smart space is a physical space (like a room) populated with intelligent systems that provide pervasive computing services to users. The central aspect of CoBrA is “[. . .] *an intelligent broker agent that maintains a share model of context for all agents, services, and devices in the space* [. . .]” [20]. This broker operates on a central server in the space. Newly acquired context is sent to this broker. The broker integrates the information to a shared context model. Context consumers can send requests to the broker, which answers these requests based on his knowledge collected in the shared context model. As stated by the authors, the “[. . .] centralized design of a broker could create a bottle neck” [21]. For that reason, the authors allow to group several brokers into so-called broker federations.

**Major contributions:** Chen et al. developed a system for supporting context awareness with the involvement of resource-limited devices. They developed a centralized approach. The central context broker is installed on a server and takes over the tasks of context acquisition and reasoning from these resource-limited devices. Context data are stored in a central knowledge base of the broker. This knowledge base consists of a context ontology providing an explicit vocabulary and allowing reasoning.

**Weaknesses:** As mentioned by the authors, the central broker also acts as a bottleneck. Even if context brokers are grouped into broker federations, the approach relies very heavily on communication. In consequence, although the authors focus on resource-limited devices, the approach requires a permanent and stable connection of these devices to an external server. Furthermore, most of the other requirements are not addressed by the approach. The context model, even though developed for exchanging information from heterogeneous devices, does not support multiple representations of the same information type. Other aspects like the context selection are not required as context information is integrated to a central knowledge base.

#### 4.1.6 CONTEXT

Chantzara et al. present in their paper an approach for evaluating and selecting context information to be used by context-aware services. This system takes the quality of information into account and automatically adapts to any changes and failures of information sources [18]. Furthermore, they propose a quality-aware discovery of service information sources which allows services to be ported easily to environments with different sets of sources. The authors motivate the need for a dynamic context selection as following:

- Context sources may provide the same type of information but vary in update frequency, accuracy, format of representation, and price.



- Context-aware services can be easily ported to other environments where different context sources provide information.
- Varying user requirements point to the provisioning of the same context-aware service with different quality levels and these levels may require different context information with different quality characteristics.

The presented approach builds on top of the IST-CONTEXT platform<sup>4</sup>, which allows to separate the logic of context-aware services from the context management functionality. So-called context brokers are acting as intermediaries between context sources and context-aware services. Context sources register the properties of the provided context information at the local context broker. Furthermore, context-aware services request context information from the local context broker. The context broker decides about the best sources to answer the context requests. If the context source is not locally available, the context broker is requesting external context brokers for the required context information. For the decision making, the context broker is calculating a utility that describes the usability of the context source in terms of accuracy, freshness, and fidelity. Besides, the selected source has to hold a cost and time response constraint. The authors mention that the context selection problem, which is according to them an instance of the Multi-Choice Multi-Dimensional Knapsack Problem (MMKP), is formulated as a utility maximization problem rather than a cost minimization problem.

**Major contributions:** Chantzara et al. provide a utility based approach for the selection of context sources, which also takes into account constraints in difference to e.g. the approach of Huebscher et al. [59] (see Section 4.1.1), which only allows to select a source without expressing constraints to limit the selection set.

**Weaknesses:** Chantzara et al. do not provide support for heterogeneous parameters and parameter dimensions. Furthermore, the selection approach does not take into account the challenges caused by the dynamic activation and deactivation of context sources. The authors also use a fixed set of quality parameters: accuracy, refresh rate, time sample, fidelity, and time response. Cost is only foreseen as a constraint, whereas it remains unclear which type of cost this parameter should be (unclear semantic: monetary or resource usage). Finally, it remains unclear if the approach supports the selection of more than one context source to fulfil more than one context request and to support sharing of context sources.

#### 4.1.7 Context Toolkit

The Context Toolkit by Salber et al. is one of the first and most referenced approaches to support the development of context-aware applications [117, 33, 32]. The basic concept of the toolkit is the reuse of specialized components for context sensing. These components are similar to graphical widgets, which are used to simplify the development of Graphical User Interfaces (GUIs). As described by Dey et al., the toolkit consists of three main abstractions [33]:

- **Widgets:** *“Just as GUI widgets mediate between a user and an application, context widgets mediate between a user and the environment.”* [33].

<sup>4</sup>CONTEXT: Active Creation, Delivery and Management of efficient Context Aware Services, IST-2001-38142-CONTEXT, <http://context.upc.es>. Last visited on Dec. 18, 2011.

- **Aggregators:** “Context aggregators can be thought of as meta-widgets, taking on all capabilities of widgets, plus providing the ability to aggregate context information of real world entities such as users or places.” [33].
- **Interpreters:** “A context interpreter is used to abstract or interpret low-level context information into higher-level information.” [33].

The toolkit is implemented as a set of Java APIs that represents its abstractions and uses the HTTP protocol for communication and XML as the language model.

**Major contributions:** The Context Toolkit was one of the first approaches to support the decoupling of context provider and context consumer. By separating the context provider into reusable components, the approach also eases the development of context-aware systems.

**Weaknesses:** The approach lacks in nearly all required capabilities: the toolkit only supports the decoupling of context provider and consumer. But it does not support any other requested feature.

#### 4.1.8 COSMOS

The COSMOS (COntext entitieS coMpositiOn and Sharing) framework is a component-based framework for managing context data in ubiquitous computing environments. The framework has been initially introduced by Conan et al. [27] and has been revised afterwards by Rouvoy et al. [113] and Abid et al. [1].

The basic structuring concept of COSMOS is the context node which “[...] is a context information modelled by a software component.” [113]. Context nodes are organized hierarchical to form context management policies. Communication in the hierarchy of context nodes may be bottom-up (notification) or top-down (observation). Context nodes providing raw context data build the leaves of the hierarchy and are called context collectors. In difference, context nodes above these leaves inference context information from the lower nodes are called context processors. To describe this composition of nodes, COSMOS provides a declarative language [113].

Abid et al. present an approach for the integration of QoS into COSMOS [1]. In COSMOS, QoC is supported through the notion of QoC operators that can integrate various kinds of QoC parameter operators, dedicated to a particular QoC parameter. A QoC-aware context node is composed of context collectors which are again context nodes and a QoC operator. A context collector collects raw metadata coming from the context sensors or sources in the distributed system. The raw metadata are then transformed by the QoC operator to deliver QoC parameters. The QoC operator is responsible for extracting required data, computing QoC, and supplying it to upper layers via the message operator. Different context collectors within the QoC operator collect the raw metadata, which get analysed by a QoC aware operator component which extracts relevant data and distributes them to QoC parameter operator components. Each QoC parameter operator computes a specific QoC parameter such as accuracy.

**Major contributions:** With its component-based and hierarchical abstraction of context providers the COSMOS framework provides a solid basis for decoupling the processes of data collection, data interpretation, and reasoning. Context providers, so called

context nodes, are encapsulated into separate components which can be composed to new context nodes. This also eases the reuse of existing nodes. An additional benefit or reason for this separation is described by Conan et al. [27] as following: *“Although adaptation actions should not be too frequent, processing context information is an activity that must be conducted more often, while data gathering is a third activity that must be continuous. Thus, we have three different activities with different frequencies. We decouple as much as possible these activities in order to obtain a non-blocking and usable framework”*.

**Weaknesses:** Despite its benefits, the COSMOS framework also has some drawbacks. The composition of context providers into context processors or reasoners has to be done with the declarative language introduced by COSMOS. This language is used to couple these nodes. As this description has to be written at design time, the system does not support the dynamic discovery of previously unknown context sources or the disappearing of context sources. This is also mentioned as future work by Rouvoy et al.: *“A second direction concerns the composition of context policies at run-time, the issue being to be able to address situations in which the intersection between these policies may be non empty, that is dynamically detecting and solving conflicts.”* [113]. Furthermore Abid et al. describe *“[...] another research direction concerns the design of look-up mechanisms to find a child in the context hierarchy with a specific QoC level.”* [1].

#### 4.1.9 EEMSS

Wang et al. developed a framework called *Energy Efficient Mobile Sensing System (EEMSS)* with the main motivation to *“[...] powering only a minimum set of sensors and using appropriate sensor duty cycles”* [144]. They developed a sensor management engine based on states and state transitions which are used to turn on respectively turned off sensors. As an example, they present an activity recognition system retrieving the user’s current activity based on data from GPS sensor, accelerometer, microphone, and a WiFi device to retrieve the surrounding WiFi. In difference to other approaches, the four used sensors are not running permanently but rather are activated only when they are useful to detect an activity change. For instance, in the state *walking* only the GPS sensor is activated. If the sensor detects that the user stopped or the sensor is not able to retrieve the current position, the WiFi device is started in order to check if known WiFi spots can be detected. At such a place the GPS is turned off and microphone and accelerometer are turned on. The accelerometer is used to detect movements of the user, whereas the microphone is used to retrieve if it is a loud or quiet surrounding.

**Major contributions:** With their approach the authors evaluated that turning off unused sensors to minimize the total power consumption increases the device’s battery life by more than 75% without waiving accuracy.

**Weaknesses:** The system created by Wang et al. has many open issues which are indispensable for ubiquitous computing systems. For example, the system can only use sensors already known at design time as these sensors are hard-coded into the XML based state description, which is used to turn the sensors on/off. A dynamic switch of sensors due to QoC issues is not possible as long it is not foreseen in the state description. For this state description, *“[...] it is important to note the system designer must be well familiar with the operation of each sensor and how a user state can be detected by a set of sensors”*.

#### 4.1.10 Gaia

Gaia is a distributed middleware system for “*Smart Spaces, which are ubiquitous computing environments that encompass physical spaces.*” [106, 110, 107]. This system also contains support for context awareness:

- “*Context Providers are sensors or other data sources of context information.*”
- “*Context Synthesizers get sensed contexts from various context providers, deduce higher-level or abstract contexts from these simple sensed contexts and then provide these deduced contexts to other agents.*”
- “*Context Consumers are entities (context-aware applications) that get different types of contexts from Context Providers or Context Synthesizers.*”
- “*Context Provider Lookup Service enables Context Providers to advertise what they offer and agents to find appropriate Context Providers.*”
- “*Context History Service lets agents query for past contexts, which are logged in a database.*”
- “*Ontology Server maintains the ontologies that describe different types of contextual information.*”

**Major contributions:** The approach adapts the idea of loosely coupling of context consumer and provider as introduced by the Context Toolkit. Additionally, they introduce a semantic annotation of the exchanged context information to address the issue of interoperability.

**Weaknesses:** The problem of selecting an appropriate context provider is passed to the different context consumers. As described by Ranganathan et al., context consumers can query the Lookup Service for a context provider that provides the contextual information they need [106]. The Lookup Service checks if any of the context providers satisfy what the consumer needs and returns the results to the application. In addition, Ranganathan et al. mention that the integration of metadata is only limited to metadata expressing the precision of the data [106].

#### 4.1.11 Hydrogen

Hydrogen by Hofer et al. is a three-layered architecture and software framework relying on a peer-to-peer network [56]. Context sources are abstracted to the so-called *Adaptor Layer*. With this layer, the simultaneous access by two or more applications to one context provider is ensured. The second layer of the architecture is the *Management Layer*, which consists of the *Context Server*. “*The context server stores all contextual information about the current environment of the device. Furthermore the context server has the possibility to share its information with other devices in range*” [56]. The third layer is the *Application Layer* consisting of the context-aware applications.

Hydrogen uses an object-oriented context model approach to realize context awareness specialized for mobile devices. Context information is stored in a context server and the management layer is responsible for providing and retrieving this context information and sharing it with other devices. Context-aware applications that use the context

information provided by the underlying layers constitute the application layer. Due to its limited capabilities a device cannot sense all the context information itself. Hydrogen provides a mechanism to share sensed context data with other nearby devices. Context data sharing is based on peer-to-peer connection over LAN, WiFi, or Bluetooth. However, the authors do not mention distributing the ‘aggregated context’, i.e., context originating from two or more devices.

**Major contributions:** The authors present an extendible framework to decouple context provider and context consumer. Context consumers can access context information from the context server either via the exchange of serialized Java objects or by XML streams.

**Weaknesses:** Even if motivating that the provisioning of meta information is an important requirement, Hofer et al. do not foresee this in their object-oriented context model [56]. The model is missing important features. For example, it does not support the expression of relationships among context information. The representations of information are not specified explicitly and the context model does not provide any semantic reference. Furthermore, the framework does not support the hierarchical composition of context providers such as context reasoners that retrieve higher-level information from raw information as requested in our requirement on loose coupling (see Requirement 2). Finally, due to the missing metadata support context selection is not possible.

#### 4.1.12 Information exchange and fusion in dynamic and heterogeneous distributed environments

Like Paspallis’ work, Reichle’s approach serves as a base for this thesis [109]. Reichle focuses on the information exchange and fusion in heterogeneous and dynamic distributed environments. He developed concepts for bridging the heterogeneity issues and considering uncertain, imprecise and unreliable sensor information in information fusion and reasoning approaches. He relies on the same context model [158] and the CQL [159]. In difference to Paspallis, he revised both context model and CQL. He extended the context model by adding support for describing imprecise and uncertain data. Additionally, the CQL, which serves originally only as description language for expressing context requests, has been extended to describe also the offered context information by context providers.

**Major contributions:** Reichle developed an approach for handling, fusion of and transforming heterogeneously represented information under consideration of their unreliability and imprecision.

**Weaknesses:** Reichle extended the context model presented by Reichle et al. [158]. However, the support for metadata in his extended model has been limited to these metadata required for his fusion approach. Furthermore, the model does not support any cost related metadata and Reichle does not support selection of context sources. His extended version of the CQL, which has been initially presented by Reichle et al. [159], does not provide any support for expressing selection criteria.

### 4.1.13 Managing Context Information in Mobile Devices

Korpiää et al. developed a “[. . .] *uniform mobile terminal software framework that provides systematic methods for acquiring and processing useful context information from a user’s surroundings and giving it to applications*” [73, 71]. They developed an ontology-based context model supporting the abstraction of raw numerical sensor data by semantic symbolic expressions [72, 70].

The presented framework basically consists of four parts:

- The blackboard-based *context manager* serves as the central communication point between the context sources and the applications.
- So-called *resource servers* abstract the actual context sources: “*The resource servers connect to any context data source and post context information to the context manager’s blackboard*” [73].
- *Context recognition services* are used to infer higher-level information from raw data, which are shared again with the applications.
- *Applications* can query for arbitrary context information from the context manager: “*The client may request context information directly from the context manager database, or specifically request context recognition*” [73].

In the context model by Korpiää et al., context information can be described by seven properties:

- The *context type* refers to a category of the context (e.g. Environment:Sound:Intensity).
- The *Context* refers to “*the symbolic ‘value’ of the information*” [72].
- The *value* is an optional property containing the numerical value.
- The *confidence* is “*an optional property of context describing the uncertainty of context. Typically a probability or a fuzzy membership of context, depending on the source.*” [72].
- The *source* property can be used to describe the source of the information.
- The *Timestamp* property contains the time of the context retrieval.
- User-defined *free attributes* can be attached to the actual information.

**Major contributions:** Korpiää et al. present an extendible framework which decouples context sources and context consumers. In addition, they provide simple support for requesting and offering semantically enriched information.

**Weaknesses:** Even if the authors provide support for the metadata type confidence to describe the characteristics of the context information, more generic support is required. With the presented model it is not possible to specify e.g. cost. Furthermore, the authors did not directly foresee an explicit semantic description of the representation of the context information (more precisely of the context value). The support for requesting and selecting context information is only based on the information type but it is not possible to take metadata constraints into account.

#### 4.1.14 MobiLife

The MobiLife project<sup>5</sup> provides a *Context Management Framework (CMF)* to realize a loose coupling of context information sources and consumers [40, 64]. *Context Sources* deliver raw context information and are wrapped into so-called *Context Providers*. These context providers can additionally include context aggregation, prediction or reasoning. *Context Consumers* are software entities that “[...] use the Context Provider’s interface as communication endpoint to obtain contextual data” [40]. *Context Brokers* are used by the context consumers to search for appropriate providers. Context providers have to advertise the provided information: “In order to achieve interoperability between Context Providers from diverse domains, the context management framework standardizes a meta model for Context Representation that all Context Providers should adhere to, in order to register themselves in a Context Broker and to enable potential Context Consumers to discover the context information they need” [40].

**Major contributions:** The framework facilitates a loose coupling of context providers and consumers. Additionally, context consumers can search and discover new and at design time unknown context providers. The concept of context broker enables the framework to access also remote context consumers.

**Weaknesses:** In order to reach interoperability, the MobiLife project decided to use standardized formats to represent context information. Even if this is a common way, it means a loss of flexibility. Furthermore, the framework does not wrap and hide context providers but rather the context broker forwards relevant advertisements of context providers to the context consumers and these consumers have to handle appearance/disappearance and selection.

#### 4.1.15 MUSIC & Paspallis

In his thesis Paspallis presents one of the bases of this work [104]. He developed a context middleware which is part of the Mobile Users in Ubiquitous Computing Environments (MUSIC) platform [96, 151].

Paspallis’ main focus is on the separation and abstraction of context provision mechanisms. As a consequence, he separates these mechanisms into context plug-ins. Context plug-ins can access either directly underlying sensors, other data sources like calendars, or also other context plug-ins. Paspallis’ approach relies on the context model by Reichle et al. [159]. Context plug-ins are queried with the help of the Context Query Language (CQL) by Reichle et al. [158].<sup>6</sup>

In order to support the hierarchical composition and the separate activation respectively deactivation of plug-ins, Paspallis developed an extended lifecycle which is inspired by the OSGi component lifecycle [102]. Plug-ins can be in different states, e.g. installed, resolved or activated. This lifecycle is reused in an extended version in our system (see Chapter 10).

---

<sup>5</sup>MobiLife (IST-511607) <http://www.ist-mobilife.org/>.

<sup>6</sup>Both documents (the one on the context model and the one on the CQL) have been co-authored by the author of this thesis and also serve as starting point for the context model respectively the context offer and query language presented in this thesis.

**Major contributions:** Paspallis decoupled context consumer and context provider. Providers are abstracted by context plug-ins. These plug-ins can be activated and deactivated.

**Weaknesses:** Paspallis uses the CQL for expressing context requests to his middleware. These requests are only matched based on the requested information type. More accurate matching and descriptions are future work. Furthermore, if more than one context plug-in exists, which fits to a context request, a plug-in is selected randomly. The support for heterogeneous representation of context information is only described based on the results of Reichle et al. [159] but has not been fully implemented and integrated in the system. For that reason, it is for example not possible to transfer context information from one representation (e.g. WGS84 coordinates) to another representation like an address.

#### 4.1.16 Nexus

The Nexus project<sup>7</sup> aims at context management for highly dynamic and complex context information in large-scale environments [9, 112, 97, 43, 79, 58].

For their system, the project developed a three-layer architecture consisting of a context information layer, the federation layer and the application layer. In the context information layer, various context data providers offer context information. As stated by Großmann et al., “[...] different context providers may provide data with different levels of detail” [43].

Federation nodes are based in the federation layer and combine context information into one global context model. When several different sensors are monitoring the same environment, inconsistencies between information provided by different context providers are unavoidable. The Nexus project tries to remove inconsistencies by integrating the provided information into a spatial world model. As described by Großmann et al., the authors take three QoC aspects, namely uncertainty, inconsistency, and trust, into account.

**Major contributions:** Nexus “[...] provides an abstraction of the single data source to the applications [...]” by integrating all available context information into a single global context model [43].

**Weaknesses:** In the Nexus project, context providers are only selected based on spatial restrictions [79] and not based on other requirements of the consumer, e.g. the requested accuracy. This is also not required as information of several providers, that offer the requested information type, are fused and then forwarded to the consumer. Besides, the project does not provide support for the dynamic activation of context providers, so it is not possible to minimize the resource consumption based on the selection.

#### 4.1.17 Quality-Aware Context Management Middleware (QCMM)

Manzoor et al. intensively discuss different aspects of Quality of Context and propose QoC metrics [86, 87, 88]. To evaluate their proposals, they implemented the so-called

---

<sup>7</sup>SFB 627: Nexus - Umgebungsmodelle für Mobile Kontextbezogene Systeme <http://www.nexus.uni-stuttgart.de/>



Quality-Aware Context Management Framework (QCMF) [87] respectively Quality-Aware Context Management Middleware (QCMM) [88]. Their system is hierarchically structured into several layers (according to the survey of Baldauf et al. [8]): sensors, context acquisition, processing, context distribution, and applications. As described by Manzoor et al., QoC parameters are used in the different layers to resolve different kinds of conflicts [87]. Hence they base their decision on QoC parameters when deciding which sensor is more reliable if two or more sensors provide the same type of information.

**Major contributions:** Manzoor et al. extensively discuss the usage of QoC and how to retrieve the different QoC parameters. In this discussion, they focus on these QoC parameters that are able to be retrieved by the middleware without relying on other parameters provided by context providers. In addition, they propose several policies to solve conflicts based on QoC.

**Weaknesses:** Even if the authors used QoC information e.g. to resolve conflicting situations during the context acquisition, the presented approach is not very generic. The focus is solely on a few QoC parameters and intensively discusses the usage of these parameters. Hence, they also provide policies to resolve conflicting situations, which are tailored to this small set of parameters. Unfortunately, the authors only implemented these policies as fixed methods to configure the middleware, instead of allowing context consumers to specify policies according to their requirements.

#### 4.1.18 Sentient Object Model

The sentient object model by Biegel et al. *“provides a systematic approach to the development of context-aware applications in mobile ad-hoc environments, supporting the important aspects of sensor fusion, context extraction and reasoning”* [13]. The model results from the CORTREX<sup>8</sup> project.

Sentient objects are used to couple the loose context providers (both sensors and other sentient objects) and context consumers (here called actuators). These objects capture the information from underlying sensors or other sentient objects and perform sensor fusion *“[...] in order to manage uncertainty of sensor data and derive higher level context information from multi-modal data sources”* [13].

**Major contributions:** The presented model decouples context consumers and providers and also allows hierarchical composition of these. The implementation is based on the STEAM (Scalable Timed Events And Mobility) middleware, which provides an event-based communication with the focus on ad-hoc wireless networks. Hence, the mobile character of users/devices in ubiquitous computing environments is explicitly addressed.

**Weaknesses:** The main focus of the approach is on fusion of information collected in a very mobile environment. Other characteristics of the information are not explicitly discussed. In addition, it remains unclear if metadata are considered during context fusion.

---

<sup>8</sup>IST-20000-26031 CORTREX - CO-operating Real-time senTient objects: architecture and EXperimental evaluation. <http://cortex.di.fc.ul.pt>

#### 4.1.19 SOCAM

The Service-oriented Context-Aware Middleware (SOCAM) by Gu et al. is a middleware “[...] for the building and rapid prototyping of context-aware mobile services.” [46, 48, 47]. The ontology-based context model is a central aspect of the middleware. The context ontology is divided into a two-level hierarchy, distinguishing between common and specific context information. The upper level describes global concepts of the ontology and the lower level is divided into several pervasive computing sub-domains, each defines specific details and properties for a particular scenario. Depending on the situation and the available devices, an appropriate sub-domain is selected from the lower level. The main focus of the SOCAM project is on context reasoning. Gu et al. distinguish between *Context Providers*, *Context Interpreter*, the *Service Location Service* and *Context-Aware Mobile Services*. Context provider “[...] abstract useful contexts from heterogeneous sources [...]”, whereas context interpreters “[...] provide logic reasoning services to process context information [...]” [47], and thus “[...] provides high-level contexts by interpreting low-level contexts.” [46]. The *Service Locating Service* serves as a central registry where context providers and interpreters have to register the provided information and where context-aware services can query for information. “Context provider can either use a service template or use OWL expressions to specify the kinds of contexts they provide. An application wishes to find out a context [...], will send a query [...] to the Service Location Service. The Service Location Service will load the context ontologies stored in the database and context instances advertised by different context providers, and apply semantic matching to find out which context provider provide[s] this context. If a match is found the reference to the context provider will returned to the application.” [46].

**Major contributions:** The authors verify that the usage of ontology-based context models is also feasible on low performance devices: they evaluated the context reasoning on a computer equipped with an Intel Celeron CPU with 600 MHz and 256 MB memory.

**Weaknesses:** SOCAM lacks supporting metadata for context information. But this is a very important requirement for context selection. Additionally, the context model does not provide any support for heterogeneous representations of context information. The architecture foresees a central server for the registry of context offers and request. But as it might not be possible to always reach a central server within every ubiquitous computing scenario, it would be better to provide more flexible support.

#### 4.1.20 Supporting pervasive computing applications with active context fusion and semantic context delivery

Roy et al. developed a resource-optimized, quality-assured framework for sensor networks [115, 114]. For this reason, they focus on fusion of context information based on Dynamic Bayesian Networks (DBN) and on ontology-based context mediation: “We propose an active context fusion technique based on DBN for fusing multimodal fragments of sensor data, and a composable rule-based mediation model to infer the situation space based on a hierarchy of contexts” [115]. During the fusion, their framework “[...] resolves information redundancy, and also ensures the conformance to the application’s quality of context (QoC) bounds based on an optimal sensor placement and / or configuration” [114].

In general, the proposed system fuses context information of two or more context sources in order to provide a certain level of quality. For this reason, the set of used context sources is selected from the set of available sources so that the potential costs are minimized but the quality requirements are still fulfilled.








Context information is described semantically by referencing to a two-level ontology similar to the ontology presented in the SOCAM project<sup>9</sup> (see Section 4.1.19). Additionally, the ontology is used to group sensor nodes providing semantically similar context data.

**Major contributions:** The authors present an interesting approach to fuse information in order to increase the provided quality. While establishing the fusion, the system tries to reduce the resource consumption.

**Weaknesses:** The general idea of the system is really interesting due to its support for cost minimization while increasing the quality of provided context information. Unfortunately, the authors did not consider several quality and cost variables. In their papers, the authors only considered “[...] update cost in terms of communication overhead [...]” [115] and the “[...] probability of correctness [...]” [114]. Even if they state that “[...] heterogeneous sensors [...] usually have different resolutions [...] accuracies, [...] data rates and formats [...]” [114], they do not provide any support to handle these characteristics. The presented selection mechanism only focuses on minimizing the cost and maximizing the accuracy to the required level. It is not possible to express context consumer-specific selection functions.

## 4.2 Summary

After the extensive discussion of the different systems, we summarize the results and provide an overview of the discussed systems. Table 4.1 lists all systems and the evaluation results regarding our previously mentioned requirements. The degree of fulfilment of a system regarding a criteria is expressed by five categories:

-  This requirement is not supported.
-  Very limited support of the system.
-  At least the most important features for fulfilling the respective requirement are provided even if some features are missing.
-  Full support of the requested feature.
-  It is not possible to justify the respective requirement.
- Both  and  are extended by footnotes to specify why the feature is not fulfilled respectively what is missing.

---

<sup>9</sup>The authors do not explicitly refer to the SOCAM project, but the described concepts look very similar and the co-author of [115] Tao Gu is the first author of the papers describing SOCAM.

Approach	Requirement 1 Discovery	Requirement 2 Loose coupling	Requirement 3 Heterogeneity	Requirement 4 Offers & queries	Requirement 5 (De-) activation	Requirement 6 Service selection	Requirement 7 Resource usage
ASC-CoOL & CoCo [134, 16, 75]	✗	✓	↗ <sup>10</sup>	↗ <sup>11</sup>	✗	✗	✗
AWARENESS [125, 123, 124]	?	?	✗	?	✗	✗	? <sup>12</sup>
CARE [3, 5, 4]	?	✓	✗	?	✗	✗	✗
CoBrA [21, 22, 19, 20]	✓	✓	✗	✗	✗	✗	✗
CONTEXT [18]	✓	✓	✗	↗ <sup>13</sup>	✗	✓	↘ <sup>14</sup>
Context Toolkit [117, 33, 32]	✗	✓	✗	✗	✗	✗	✗
COSMOS [27, 113, 1]	✗	✓	✗	↘ <sup>15</sup>	✗	✗	✗
EEMSS [144]	✗	✗	✗	✗	✓	↗ <sup>16</sup>	✓
Gaia [110, 106, 107]	✓	✓	↗ <sup>17</sup>	↗ <sup>18</sup>	✗	✗	✗
Huebscher et al. [59]	?	✓	✗	✓	✗	↗ <sup>19</sup>	↘ <sup>20</sup>
✗ = no support   ↘ = very limited support   ↗ = partial support   ✓ = full support							

<sup>10</sup>The metadata representations are fixed.

<sup>11</sup>The CoCo system requires the context consumers to express required meditation tasks by themselves instead of only specifying the required information. Furthermore, no dynamic selection and according specifications in the requests are supported and the usage of QoC parameters to state more precisely the context request is not possible.

<sup>12</sup>This issue has been motivated but not addressed in the different papers.

<sup>13</sup>Cost is only implicitly taken into account during selection as it is possible to specify one cost constraint and ergo to limit the search space.

<sup>14</sup>Cost preferences are limited to be expressed by only one cost constraint. However, the system does not take the cost parameter into account during the selection.

<sup>15</sup>Context queries contain explicit references to context sources instead of criteria to find these sources.

<sup>16</sup>Selection based on state diagram; no explicit support for selection of context provider from a set of services providing the same type of information.

<sup>17</sup>Heterogeneous representations are only implicitly described by the context type; no support for transformation between representations.

<sup>18</sup>No support for metadata constraints and selection functions.

<sup>19</sup>The selection is only based on a fixed set of quality parameters. Cost parameters or other quality parameters as the mentioned ones are not considered.

<sup>20</sup>The issue is only addressed implicitly by minimizing the number of used context sources. It is not possible to select for example a cheap context source as cost parameters are not considered.

Approach	Requirement 1 Discovery	Requirement 2 Loose coupling	Requirement 3 Heterogeneity	Requirement 4 Offers & queries	Requirement 5 (De-)activation	Requirement 6 Service selection	Requirement 7 Resource usage
Hydrogen [56]	✗	✓	✗	✗	✗	✗	✗
Korpiää et al. [73, 72, 70, 71]	✓	✓	✗	↔ <sup>21</sup>	✗	✗	✗
MobiLife [40, 64]	✓	✓	✗	✓	✗	✗	✗
MUSIC & Paspallis [104]	✓	✓	↔ <sup>22</sup>	↔ <sup>23</sup>	✓	✗	↔ <sup>24</sup>
Nexus [9, 112, 97, 43, 79, 58]	✗	✓	✗	✗	✗	✗	✗
QCMM [86, 87, 88]	?	✓	✗	↔ <sup>25</sup>	✗	↔ <sup>26</sup>	✗
Reichle [109]	✓	✓	↔ <sup>27</sup>	↔ <sup>28</sup>	✗	✗	✗
Roy et al. [115, 114]	✓	✓	✗	↔ <sup>29</sup>	✗	↔ <sup>30</sup>	↔ <sup>31</sup>
Sentient Object Model [13]	✗	✓	✗	✓	✗	✗	✗
SOCAM [46, 48, 47]	✓	✓	✗	✓	✗	✗	✗
✗ = no support   ↔ = very limited support   ↔ = partial support   ✓ = full support							

**Table 4.1:** Overview on Existing Context-aware Systems

It can be seen that, none of the existing systems fulfil all requirements, whereas most of the approaches decouple context provisioning and context consumer and support semantic discovery of new context providers. Only few approaches cover the other requirements.

<sup>21</sup>No support for constraints or selection functions.

<sup>22</sup>Not fully implemented for context information; no support for multiple representation of metadata.

<sup>23</sup>Missing support for e.g. context selection.

<sup>24</sup>Support only implicitly through sharing of context sources.

<sup>25</sup>No support for selection properties.

<sup>26</sup>Selection based on middleware policy; limited set of used parameters for selection.

<sup>27</sup>No support for multiple representation of metadata.

<sup>28</sup>Missing support for e.g. context selection.

<sup>29</sup>Missing generic support for arbitrary cost and metadata constraints.

<sup>30</sup>No generic selection functions.

<sup>31</sup>Limited to one cost dimension.

The systems by Strang et al. [134, 16, 75], MUSIC & Paspallis [104], and Reichle [109] provide initial support for the conversion of context information from the provided representation into the requested representation. However, none of these approaches provide this support also for metadata of context information. In addition, both Strang et al. and Paspallis do not support the automatic establishment of transformation chains, which is supported by Reichle [109].

Huebscher et al. [59] and the MobiLife system [40, 64] provide the only systems supporting the specification of context offers and needs including metadata constraints and preferences used for the selection of context sources. Additionally, the system by Huebscher et al., Chantzara et al. [18], EEMSS [144], QCMM [86, 87, 88], and Roy et al. [115, 114] are context-aware systems providing support for context selection. From this set the approach by Chantzara et al. [18] has to be highlighted as it allows a selection of context sources based on generic metadata.

The dynamic activation and deactivation is an important requirement in order to minimize cost respectively resource consumption. The systems by MUSIC & Paspallis [104] and EEMSS [144] are the only approaches with concrete support for this feature.

Finally, the explicit minimization of the total amount of used resources is only fully addressed by EEMSS [144]. Whereas e.g. MUSIC & Paspallis [104] or Huebscher et al. [59] support this feature only implicitly by deactivating unused context sources. But they do not take into account cost information to select which source to use. Chantzara et al. [18] consider at least one cost parameter (with unclear semantic) during the selection. In difference to these other approaches, EEMSS [144] explicitly focuses on minimization of resource cost and also demonstrates that their system reduces the resource usage.

We will see in the next section that we reuse some of the features provided by the discussed solutions, especially by CoOL [134] and the systems of Paspallis [104] and Reichle [109]. In our approach, the issue of dynamic selection and activation respectively deactivation is explicitly addressed. Context services are described by an extensive context offer and query language. Mediator chains are automatically established to transform the context information into the requested format. In order to make a precise selection of deactivated context providers, historical values are used to estimate the current status of the deactivated context providers (more precisely the QoC parameters).

**Part II**

**Solution Approach**





## 5 Overview

---

*Any sufficiently advanced technology is indistinguishable from magic.*

– Arthur C. Clarke (1917-2008)

Clarke's third law. Profiles of The Future (1961)

Modern smartphones contain a wide range of different *sensors* and various kinds of other *data sources* like calendars which are also useful to retrieve context information. Additionally, *reasoning*, *fusion*, and *inferencing mechanisms* can be used to retrieve high-level context information from other context information. In this work, we abstract all these context providers as *context services*.

While on the one hand several context services can exist, on the other hand, context-aware applications can request different kind of context information. It is also common that a single application requests for several different information types. These requests are called *context requests* or *context queries*<sup>1</sup>. A *context consumer* is a small part of an application, which queries and receives the required information of one type, thus for a single context request.

Applications and the according context consumers build the top layer of our system. Context services and the encapsulated sensors, databases, reasoning mechanisms build the bottom layer as depicted in Figure 5.1. The middle layer of our system contains our *context middleware*. Context services register their offered information with a so-called *context offer*. Similar context consumer express *context queries* in order to retrieve context information from the middleware. A context middleware instance can communicate with other middleware instances and also with servers to exchange context offers and also context information<sup>2</sup>.

As shown in Figure 5.1 we can distinguish between several stakeholders in our system. *Database developers*, *context sensor developers*, and *context reasoner developers* are responsible for the functionalities of the databases, sensors, and reasoning mechanisms used as context sources, respectively. Especially the database developers and context sensor developers are mostly developers of the device manufacture (e.g. Samsung<sup>3</sup> or Apple<sup>4</sup>) or of the operating system (OS) of the device (e.g. Android<sup>5</sup> or iOS<sup>6</sup>)

<sup>1</sup>The terms 'context request' and 'context query' are used interchangeable in this thesis.

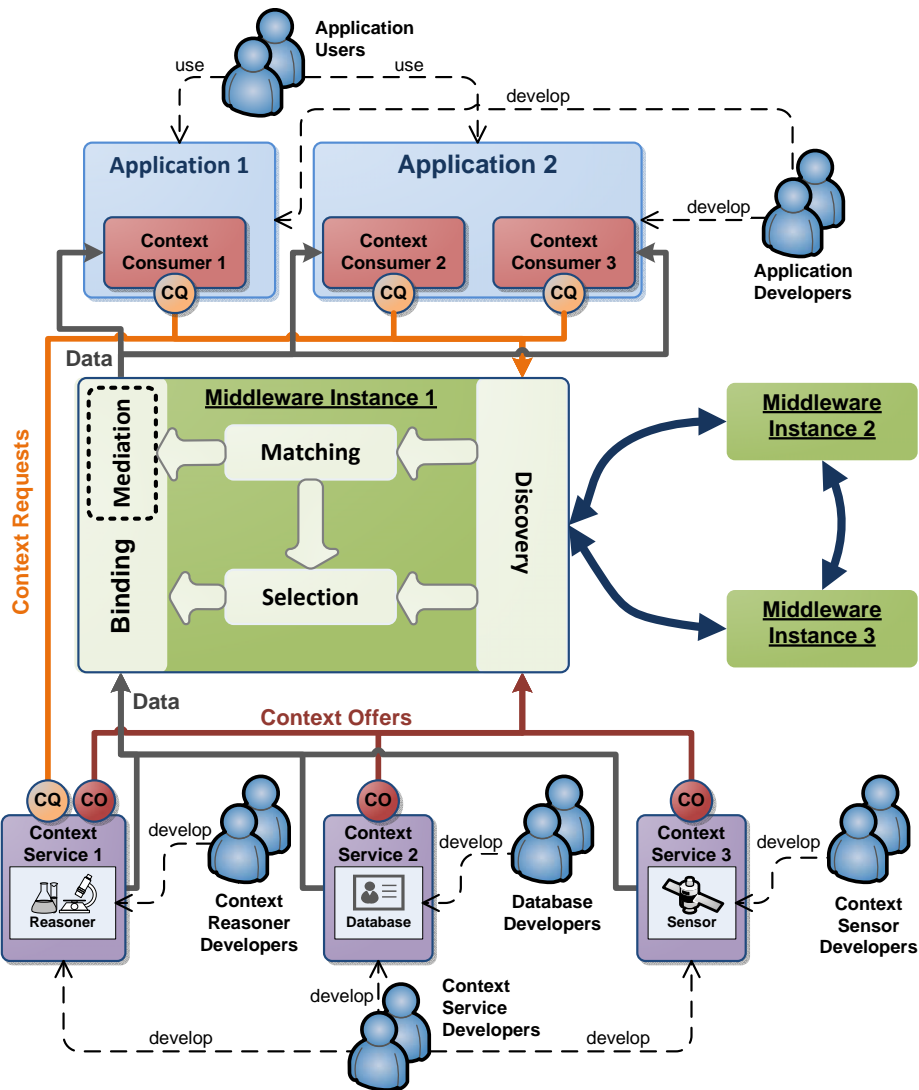
<sup>2</sup>It depends on the implementation if the middleware communicates only with other middleware instances, directly with other (remote) context services and/or other external servers. Communication and also discovery aspects are encapsulated in separate binding respectively discovery plug-ins, which can be independently developed and accessed. This is discussed in more detail in Chapter 10.

<sup>3</sup><http://www.samsung.com>

<sup>4</sup><http://www.apple.com>

<sup>5</sup><http://www.android.com>

<sup>6</sup><http://www.apple.com/iphone/ios>



**Figure 5.1:** Detailed Overview of the Solution Approach including Stakeholders.

The *Context Service Developers* encapsulate the provided information sources by mainly writing a proxy component to translate the database or sensor API to the Context Service API. Furthermore, they write the context offer describing what context information is provided (which type, regarding which entity, in which representation), in which quality and to what cost the information can be provided. Context service developers and for instance context sensor developers can also be one and the same person. In the following, our focus is mainly set on context service developers.

For applications we differentiate between the typical two roles of *Application Users* and *Application Developers*. The application users are the end-user and have no influence on the development of the actual application. Nevertheless, the application developer can offer ways (e.g. by user settings or explicit dialogues with the user) to influence the context requests and as a result the selection of context sources. This is for example useful as users might have different opinions regarding cost-minimization.

The following chapters give a detailed description of the core functionalities of our middleware as indicated in Figure 5.1. These functionalities are required to combine the loosely coupled context services and context consumers. As a cross-cutting concern, we will first introduce our context model. The context model and the corresponding data descriptions build the basis for all other components of our work. For example, the context requests and offers are described in our *Context Offer and Query Language*, which inherits concepts from the context model. This language will be described in Chapter 7. The context offers and queries are registered to the **Context Discovery**. As the discovery component does not contain any specific new technology, we will not focus on that. It will rather be discussed with the rest of the overall architecture in Chapter 10.

After registering context offers and requests, the middleware starts the **Context Matching** and verifies which offers can be used as sources for the different requests. During this matching process also the tentative **Context Mediation** is evaluated. Context mediation might be necessary to bind specific context sources to context consumers which do not provide the information in the request representation. Context matching and mediation are discussed in detail in Chapter 8. Finally, the **Context Selection** decides which set of context offers should be activated. The selection algorithm tries to fulfil the different preferences of the context consumers as best as possible and additionally tries to minimize the number of used resources (see Chapter 9).

The resulting middleware architecture and further implementation details are discussed in Chapter 10.



## 6 Context Model

---

*The nice thing about standards is that  
you have so many to choose from.*

– Andrew S. Tanenbaum (1944-)  
Computer Networks (1981)

A context model contains an arbitrary number of context artefacts and provides information about these artefacts as well as their representations and semantics.

The usage of the context model in a Ubiquitous Computing (UC) environment causes additional requirements: context information is distributed on an arbitrary number of devices; these devices are mobile and can appear and disappear; we cannot assume a UC environment to contain only homogeneous devices, these are rather heterogeneous providing different sets of context artefacts in different representations and under different names. This is also described in Requirement 1 ‘Semantic discovery and integration of independently developed context services and consumers’. The context model provides a baseline for a solution of this requirement. Additionally it directly supports to solve Requirement 3 ‘Exchange and interpretation of heterogeneously presented context information’ (see Section 1.2).

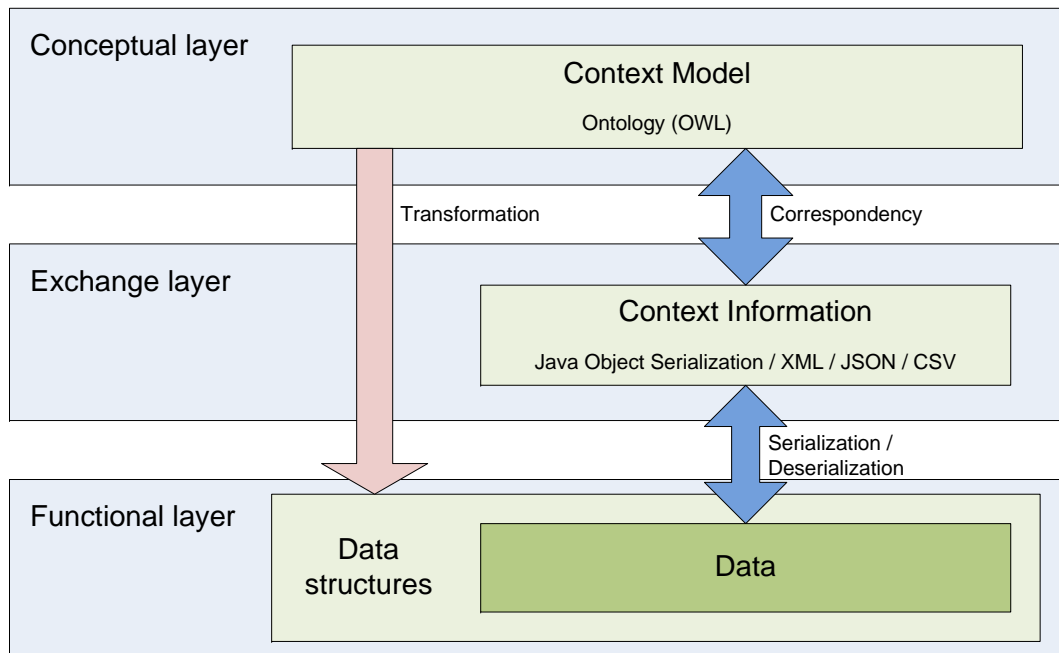
Furthermore, it takes the general characteristics of context data into account, like ambiguity, impreciseness or incompleteness (see Section 2.2.2). These characteristics are expressed as metadata which require generic support to describe arbitrary quality and cost information. This support is not only useful to describe the specific characteristics but also to provide an input for the selection process as requested by Requirement 6 ‘Dynamic selection of context services based on QoC and CoC’.

The context model forms the baseline of our approach. The initial version of this context model has been presented among others by the author of this thesis in [158]. Reichle developed this initial version further but focused on support for context fusion [109]. Reichle’s approach mainly supports metadata required for context fusion, e.g. information reliability, precision, uncertainty and outdateness. In this work, we adapt the extensions by Reichle but significantly extend the support for metadata (Section 6.3) and operations to calculate the metadata (Section 6.4). In difference to Reichle, the focus here is not on specific metadata but rather on generic support for arbitrary metadata and cost information.

### 6.1 Layers of the Context Model

A context model in a ubiquitous computing environment has also to fulfil some requirements with regard to abstraction of information, its representation, and finally

the concrete implementation. Firstly, it has to provide a common vocabulary to achieve interoperability between heterogeneous context services and consumers. Secondly, during runtime, especially on resource restricted mobile devices, a well-organized slim data structure containing the context data is necessary and thirdly, this data has to get exchanged between the different nodes in the UC environment. Based on this motivation, we have identified three abstraction layers of the context model. As shown in Figure 6.1, we can distinguish between three layers: the conceptual layer, the exchange layer, and the functional layer.



**Figure 6.1:** Layers of the Context Model

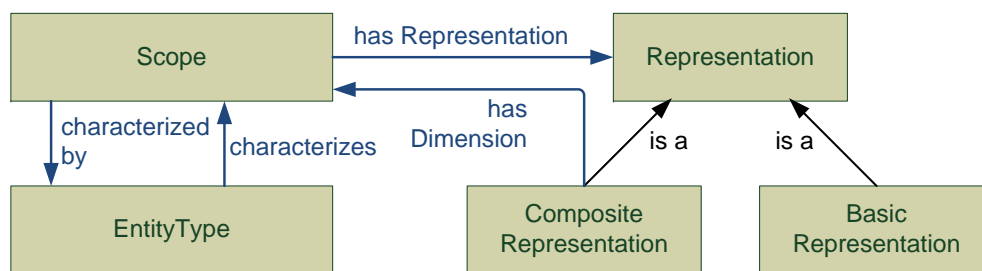
The *conceptual layer* consists of the context ontology specified in OWL 2 [139, 140] (see Section 3.3.2 for more details). The context ontology provides a common vocabulary including the semantics as well as the structure of the context information. Furthermore the ontology consists of static context information, like the relationship status of persons, which generally do not change very often. The context ontology is defined at design time and transformed into the data structures of the *functional layer*. These data structures hold the context information at runtime. The third layer is the *exchange layer* containing the serialized context information. This context information refers to the corresponding concepts of the context ontology and is serialized from respectively deserialized into the generated data structures. The serialized context information is used for the exchange of context services and context consumers.

In the following sections we will focus on the core concepts of the context model, which are mainly described in the context ontology. These concepts also correspond to the other layers due to the relationship of the layers.

## 6.2 Context Information

According to our definition (see Definition 2.1), context is defined as every information that can be used to characterize the situation of an entity. While formalizing this definition, we realized that several context information can be of the same information type but differ in its representation. For example information regarding the position can be represented as World Geodetic System 1984 (WGS84) coordinates or an address. This information type is called the context *scope*. Corresponding to our definition and as shown in Figure 6.2, this scope characterizes a certain *entity*, e.g. a person or an object like a device.

Strang et al. [134] introduced in their *Aspect-Scale-Context (ASC) model* the concepts of aspects, scales, and context information as following: “Each aspect aggregates one or more scales, and each scale aggregates one or more context information.” We adopt this model but adjust it to our purposes.

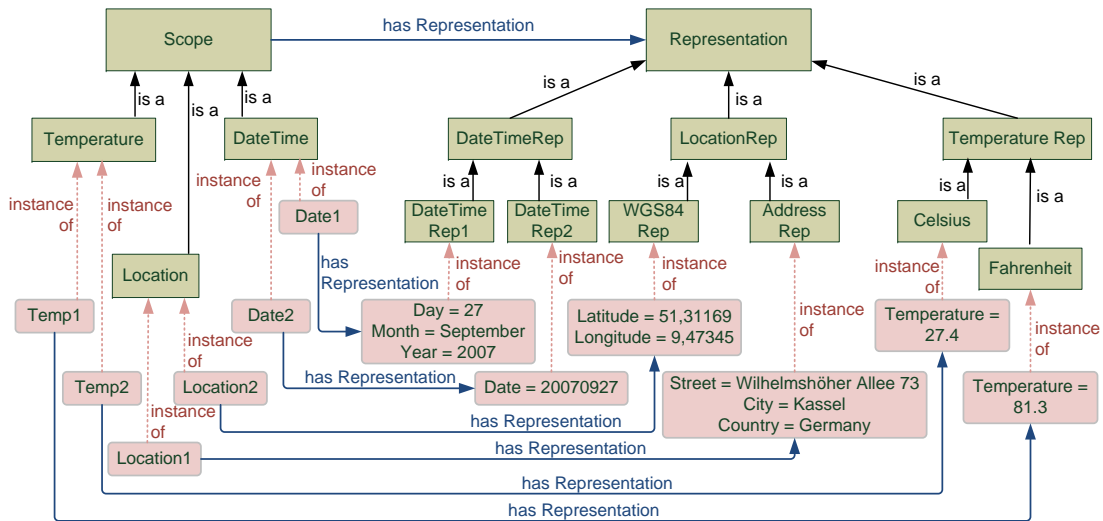


**Figure 6.2:** General Concepts: Entity, Scope and Representation

A scope can have several *representations*, e.g. a position can be represented as WGS84 coordinate or as an address. Figure 6.2 shows that these representations can be either atomic/basic or composite representations. A basic representation consists of a single value like an Integer, Double or String and optionally also a unit. A composite representation consists of several dimensions. Likewise, these dimensions are scopes which can also have several representations. For example, an representation ‘WGS84 coordinate’ consists of the two dimensions with the scope ‘longitude’ and ‘latitude’. These two dimensions have again a representation, like. e.g. a basic representation consisting of a Double value and a unit ‘mm’.

By supporting various representations for a certain scope, we directly address the interoperability of context information. It would be rather restrictive to fix the representation e.g. of a position to WGS84 coordinates as it depends on the application and its users in which format the expected information are represented<sup>1</sup>. A position in WGS84 coordinates is e.g. easier to use within an application to determine all objects with a certain radius whereas a position in the address format is more human-readable than a position in coordinates.

<sup>1</sup>It is of course the developer of the context service who appoints the representation of the provided information. In difference, the application developer of a context-aware application decides on the requested representation. However, the application developer can also offer the option for application users to select the representation. This is not useful for the actual application logic but for information presented within the user interface, for instance to allow the user to select his preferred representation for a distance (meter vs. mile) or temperature (Fahrenheit vs. Celsius).



**Figure 6.3:** Several Examples for Multiple Representations

Figure 6.3 shows several examples for multiple representations. To improve the readability, the different dimensions of a representation are summarized in one block, e.g. the *DateTimeRep1* consists actually of the three dimensions *Day*, *Month* and *Year*. As depicted, representations can vary in several ways: variation in format (like *Date1* and *Date2* in the example), variation in value/unit (like *Temp1* and *Temp2*) and variation in both format and value/unit (like *Location1* and *Location2*).

To ensure the exclusive usage of compatible representations for a certain scope, it is possible to limit the range of the *hasRepresentation* property of this scope, e.g. for the scope *Location*:

$$\boxed{\textit{hasRepresentation} \textbf{ only } \textit{LocationRep}} \quad (6.1)$$

Without this constraint, it would be possible to combine incompatible scope and representations, like the scope *DateTime* and the *TemperatureRep* representation *Fahrenheit*.

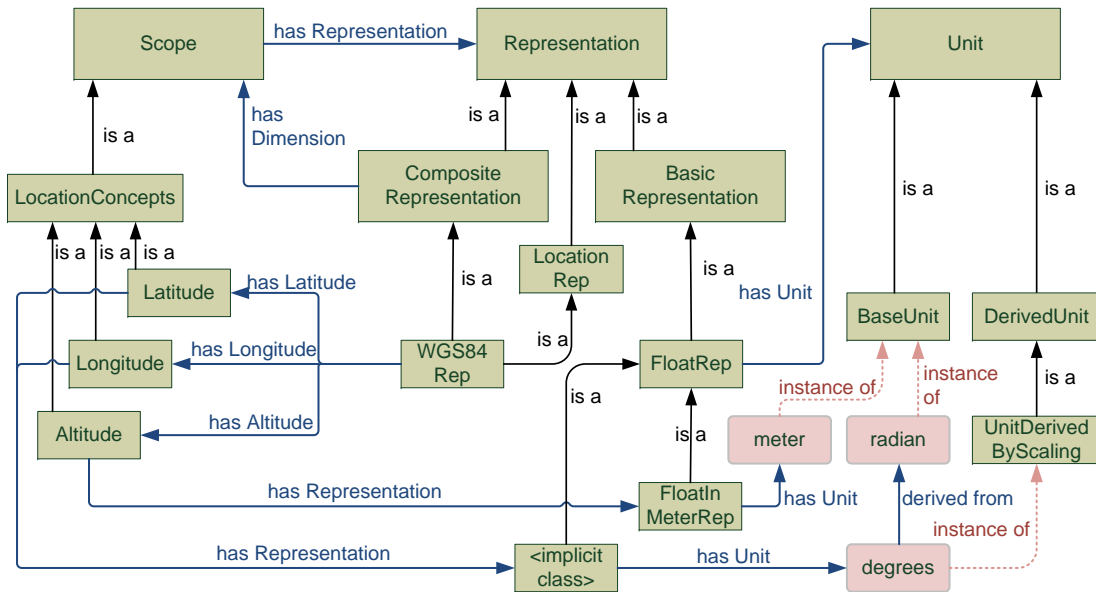
To get a deeper understanding of a composite representation, the *WGS84Rep* is depicted in more detail in Figure 6.4. A composite representation consists of several dimensions, which are again scopes. This is indicated by the ‘*has Dimension*’ property. To model a specific representation like the *WGS84Rep*, this property has to be limited to the actual dimensions. In the depicted example, these dimensions are *Latitude*, *Longitude*, and *Altitude*. In our OWL 2-based implementation, this can be done in two ways:

- by limiting the *range* of the object property ‘*hasDimension*’, e.g.

$$\boxed{\textit{hasDimension} \textbf{ only } \textit{LocationConcepts}} \quad (6.2)$$

- or by adding sub/child object properties of the object property ‘*hasDimension*’ to the ontology, e.g. the subproperty ‘*hasLatitude*’ with the *domain* ‘*WGS84*’ and the *range* “*Latitude*” as depicted in Figure 6.4.





**Figure 6.4:** Detailed Example for a Composite Representation

The first option has the drawback, that it is necessary to define additional constraints to limit the multiplicity. For example, additional constraints are required to express that a WGS84 position consists exactly of one latitude and one longitude, whereas the altitude is optional. For this reason the second option is preferable.

Additional to expressing the dimensions of the representations, it has to be further specified which representation these dimensions should have. Otherwise the dimensions would be provided in arbitrary representations. In the example depicted in Figure 6.4, latitude and longitude should be expressed as *Float* values with the unit *degree* and the altitude should be expressed as *Float* with the unit *meter*. The class *Unit* is based on the NASA SWEET Units ontology [135]. This ontology allows a conversion between units only based on the information specified in the ontology. The SWEET ontology contains the classes *Prefix* and *Unit* with its subclasses *BaseUnit* and *DerivedUnit* and several object properties like *hasPrefix* or *derivedFrom*. The concrete units are individuals of the different classes, e.g. *meter* is an individual of the class *BaseUnit*, whereas *degrees* is an individual of *UnitDerivedByScaling*, which is again a subclass of *DerivedUnit*. This individual *degree* is derived from the base unit *radian* by scaling. To limit the *hasRepresentation* property of the dimension scopes, there are again two different ways. In both cases, constraints are used to limit the range of the *hasRepresentation* property.

- In the first variant, the range of the *hasRepresentation* property is limited to an existing subclass of the *Representation* class, e.g. in our example the dimension *Altitude* is limited to the *FloatInMeterRep* by

$$\boxed{\textit{hasRepresentation} \textbf{ only } \textit{FloatInMeterRep}} \quad (6.3)$$

This representation again has a constraint expressing that limits the *hasUnit* property to the unit individual *meter*: *FloatInMeterRep* by

$$\boxed{\textit{hasUnit} \textbf{ value } \textit{meter}} \quad (6.4)$$

- In the second variant, an implicit class is created by first limiting the *hasRepresentation* property of the dimension class and within this constraint further limiting the *hasUnit* property of the representation, e.g.

$$\boxed{\textit{hasRepresentation} \textbf{ only } (\textit{FloatRep} \textbf{ and } (\textit{hasUnit} \textbf{ value } \textit{degrees}))} \quad (6.5)$$

This ends in a new implicit class, as depicted in our example for the dimensions Latitude and Longitude.

Both ways can be used without any constrictions as they finally end in the same result. Reichle decided on the usage of implicit classes [109]. This variant has the benefit that the ontology is rather small and clear but requires a more accurate and complex specification of the dimension restriction. We support both ways and it is up to the developer which way he prefers.

### 6.3 Metadata

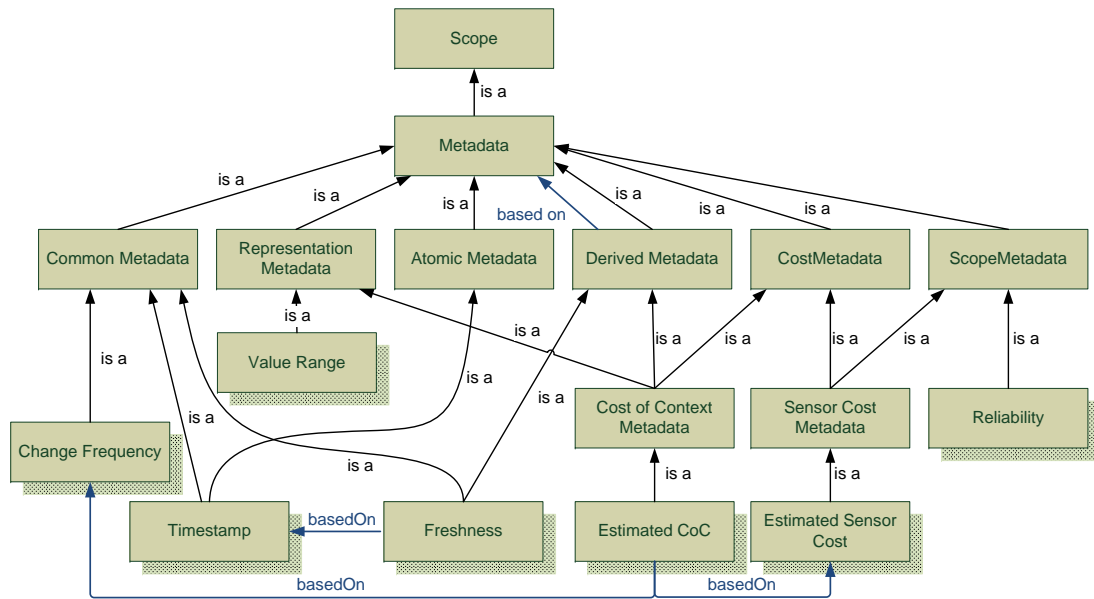
As mentioned by several authors, context information can be unreliable, inaccurate, outdated, etc.<sup>2</sup>. Metadata are important information about the actual context information and allow to handle these characteristics of context information. As aforementioned, our context model relies on the work of Reichle et al. [158] and its successor [109]. Whereas in the initial version, metadata only had a marginal position at best, in the successor metadata played a more important role as they are indispensable for Reichle's context fusion approach [109]. However Reichle focused on specific metadata required for his concept<sup>3</sup>. In our work, metadata are mandatory criteria for selecting context services according to requests. For this reason, we do not rely on any specific metadata. Instead we provide a rather generic approach to support arbitrary cost and quality data.

In our context model, metadata are also again scopes which can have several representations. For example, it is possible to describe the accuracy of a position as a derivation in meter or in millimeter. In difference to our work, Reichle [109] only links metadata describing impreciseness with the representation of context data, whereas the representations for other metadata are fixed.

In Definition 2.3 we defined Quality of Context (QoC) as any information that describes the quality of information that is used as context information. Furthermore, we defined in Definition 2.4 Cost of Context (CoC) as a parameter associated to the context that indicates the resource consumption used to measure or calculate the piece of context information. Even if parameters of both classes can be generally defined as metadata and are modelled equally, these parameters are handled separately during the selection (see Chapter 9). For that reason, it is not possible to completely resign the distinction but rather to abstract as much as possible – but still be able to identify which metadata are cost related and which are quality related. For this purpose, we introduced the class *Cost Metadata*. Instances of this class or of its subclasses are related to cost, whereas all other instances are related to quality.

<sup>2</sup>A detailed discussion on the different characteristics of context data can be found in Section 2.2.2.

<sup>3</sup>For example, Reichle does not have any use for metadata related to cost and does not support them.



**Figure 6.5:** Context Metadata

In general, metadata can be classified into two subclasses. The first subclass contains metadata describing the information source, like reliability or the sensor cost. The second class consists of metadata which are associated to, or respectively depend on the representation of the context information, e.g. the value range of such a representation or also the cost of context information. Additionally there are metadata which can be used to describe both aspects, information source and representation. For example, a timestamp can be associated to a scope to express when the a certain piece of context information has been gathered by the context service whereas a timestamp for the representation expressed when the context information has been transformed into this representation. As depicted in Figure 6.5 we distinguish between *scope metadata*, *representation metadata*, and *common metadata*, which can be both scope and representation metadata. *Scope metadata* describe rather metadata related to the information source. *Representation metadata* on the other hand depend on the representation of the context information and characterize the associated data.

Cost related metadata are metadata which are instances of the *cost metadata* class. As described, these metadata can be associated to either the representation or the scope. Cost metadata associated to the representation are used to express the ‘Cost of Context’: the cost which originated from the calculation of a certain piece of context information. Consequently, we introduced the class *cost of context metadata* which is a subclass of both classes *representation metadata* and *cost metadata*. In difference to cost of context metadata, metadata can exist that describe e.g. the average cost consumption of the context service. For that reason, these metadata are not related to a certain piece of context information. They are collected in the class *sensor cost metadata* which is a subclass of *cost metadata* and *scope metadata*.

It is clear that various metadata have a correlation to other metadata. For example in order to calculate the *freshness* (see Section 2.2.2) of context information it is necessary to know its date of origin, e.g. in form of a *timestamp*. Furthermore, it is possible to

retrieve the estimated cost for a single piece of context from the estimated sensor cost if the frequency of data acquisition is known. Consequently, metadata (independent if cost related metadata or quality related metadata) can be either atomic or derived.

*Atomic metadata* are these metadata which have to be calculated directly by the information source or the information processor, e.g. the process transforming the information into the requested representation. In difference, *derived metadata* can be derived from other metadata. Figure 6.5 contains the metadata class *Freshness*. This class is a subclass of *common metadata* but also of *derived metadata*. As aforementioned, all derived metadata rely on one or more other data, e.g. the freshness of context information can be calculated from the timestamps of these information. This is indicated through the OWL object property *basedOn* directed from the freshness class to the timestamp class.

The previously introduced *cost of context metadata* class is also a subclass of the *derived metadata*, e.g. the *estimated cost of context* can be calculated based on the *change frequency* of the sensor and the *estimated sensor cost*.

Nevertheless, we only provide a generic frame for the different types of context metadata but do not offer an exhaustive set of metadata. The ontology contains an appropriate set of metadata to demonstrate all concepts within this work, but it is easily extendible as presented in Section 6.5.

## 6.4 Operations

In the previous sections we described our support to allow arbitrary representations for context information and also for its metadata. Furthermore, we described that metadata can be derived from other metadata. This flexibility requires support for example to provide information in requested representations or to perform the mentioned metadata calculations.

Operations in general are calculations specified in the ontology to produce or transform context information or its metadata based on other information or metadata. As depicted in Figure 6.6, we distinguish between *Inter-Representation Operations (IROs)* and *Metadata Operation*. The concept of Inter-Representation Operations has already been introduced by Reichle et al. [158]. These IROs can transfer information from one representation to another representation. With the concept of IROs as introduced by Reichle et al. it is not possible to derive metadata from other metadata. Inter-Representation Operations transform context information of one scope from a representation into another representation of the same scope. But in order to derive metadata, this concept has to be extended to transfer information of one scope in a representation to another scope in another representation. This support is provided by the concept of *Metadata Operations*. Metadata Operations are used to calculate the previously introduced derived metadata. As both Metadata Operations and Inter-Representation Operation have a common subset of properties, we generalized these classes and introduce the class *Operation*.

As shown in Figure 6.6 an operation works on a certain scope, provides its information in a certain representation (indicated by the *hasOutput* property), and has a grounding.

The grounding contains information of how to invoke the specific operation, e.g. by calling a web service or by starting a certain OSGi bundle [102].

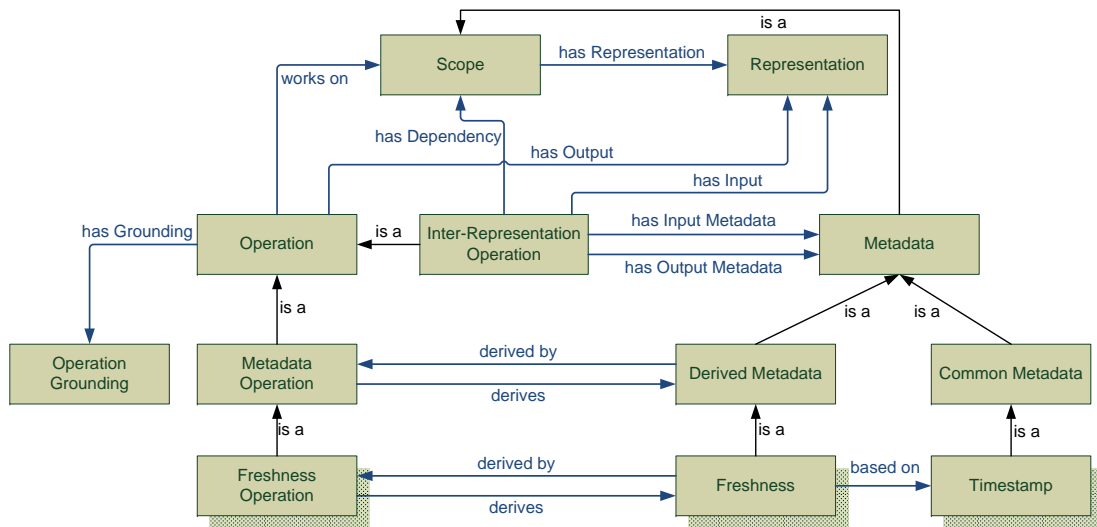


Figure 6.6: Context Operations

In the ontology, *Operations* in general and hence also its specialisations *Inter-Representation Operation* and *Metadata Operations* are specified as classes. Intuitively, adding a new concrete IRO or metadata operation would mean to create a new individual of the IRO class respectively the Metadata Operation class. But this would also mean that the operation would only be applicable for one single individual of one specific scope and one single individual of the corresponding representation. In consequence, also concrete (Inter-Representation or Metadata) operations have to be specified on class level by the usage of constraints limiting the different object properties, for example:

$$\boxed{\textit{hasOutput} \textit{ only AddressRep}} \tag{6.6}$$

In difference, a new individual of the *Operation Grounding* class has to be created to add a new grounding to such a concrete operation. Additionally, an individual of the previously added operation class (describing the concrete operation) has to be added to establish the grounding. This new individual does not require any additional information, for instance regarding the representation as this information is already available within the class definition.

The classes *Inter-Representation Operation* and *Metadata Operation* are both specialisations of the *Operation* class and inherit the above mentioned properties and associations.

### 6.4.1 Inter-Representation Operations

In order to support interoperability of context information, we developed the concept of *Inter-Representation Operations* based on the ideas of Strang et al. [134]. Inter-Representation Operations convert context information of a certain scope from one representation into another representation, like a position represented in WGS84 coordinates into a position represented as an address.

As shown in Figure 6.6, an IRO is a specialization of an operation working on a certain scope like ‘position’, provides the information in a certain output representation like an address, and can have several groundings for instance to web-services.

Additionally to the *Operation* class the *Inter-Representation Operation* class requests its data in a certain input representation (by the OWL object property *has Input*, e.g. WGS84 coordinates).

IROs can also influence the metadata of the context information, for example the accuracy if the address for a certain coordinate is unknown. Hence, an IRO also has additional requirements on input and output metadata, as it influences and/or calculates these metadata. For example, the transformation of a position represented in WGS84 coordinates into an address can influence the accuracy of the position information. For these purposes, the IRO class has the OWL object property *has Input Metadata* and *has Output Metadata* associating the IRO class with the *Metadata* class, as shown in Figure 6.6. To manage multiple representation of metadata, the properties *has Input Metadata* and *has Output Metadata* also require information in which representation the metadata are expected respectively provided. This can easily be done within the ontology by limiting the range of the particular property. For example, to express that an IRO expects the input metadata *Freshness* to have the representation *DoubleRep* and the Unit *Second*, the following expression can be used:

$$\boxed{\textit{hasInputMetadata} \textbf{ only } (\textit{Freshness} \textbf{ and } (\textit{hasRepresentation} \textbf{ only } (\textit{DoubleRep} \textbf{ and } (\textit{hasUnit} \textbf{ value } \textit{Second}))))} \quad (6.7)$$

Additionally, IROs can have dependencies to other context information, e.g. to calculate an absolute position of a person located in a building based on relative position of this person within this building (e.g. the room number), the absolute position of the building is required. The exact specification of the requested information works similar as for the specification of the input and output metadata by limiting the range of the OWL object property.

During the implementation, it turns out that it is sometimes not applicable to specify all functionalities of an IRO in the ontology. The main reason is, that IROs exist that are applicable in a wide range of scopes and representations (e.g. an IRO applicable for most of the atomic representations to cast between them) and it would cause a big overhead to specify all ranges. For this reason, we introduced subclasses of the IRO class: *Specific IRO* and *Generic IRO*. Both classes do not provide any additional properties to the inherited ones. But whereas all IROs of the *Specific IRO* class have to specify the provided and requested representation and scopes explicitly, the IROs of the *Generic IRO* class use in their specification more generic scopes and representation. For example, an IRO which can cast between different simple data types, like Integer and Double, can be specified to work on every scope and every atomic representation. To determine whether this IRO is useful to convert data from one representation to another, it has an extended interface compared to the IROs of the *Specific IRO* class. This extended interface contains a method which can be called to check whether it is applicable or not. Such support also increases the ease of reusing existing implementation. Most unit conversion implementations are a good example. They already provide support to check whether a conversion is feasible or not.

## 6.4.2 Metadata Operations

*Metadata operations* are used to calculate the derived metadata. This is indicated by the *derives* property shown in Figure 6.6. It is the only additional property of the *metadata operation* to the inherited properties *works on*, *has Grounding* and *has Output*. The depicted example shows a generic freshness operation, which derives the metadata of the type freshness from the timestamps. This generic freshness operation must have at least one subclass which has additional concrete specifications of the required input and output representations. For example the freshness operation can have a specialization which calculates the freshness as a Double value in seconds based on a Unix timestamp<sup>4</sup>. The output representation can be specified in OWL 2 as following:

$\text{hasOutput \textbf{only} (DoubleRep \textbf{and} (hasUnit \textbf{value} \textit{second}))}$	(6.8)
--	-------

The input representation is specified as:

$\text{worksOn \textbf{only} (Timestamp \textbf{and} (hasRepresentation \textbf{only} \textit{Time\_UnixTimeStamp}))}$	(6.9)
--	-------

Here, the output representation refers to an implicit class, similar to the examples shown and explained in Figure 6.4. In difference, the input representation in our example is an explicit/concrete representation. It is also possible to refer to an implicit class, for example by referring to the *FloatRep* class and to add a further constraint limiting the *hasUnit* property.

## 6.5 Hierarchical Composition of the Ontology

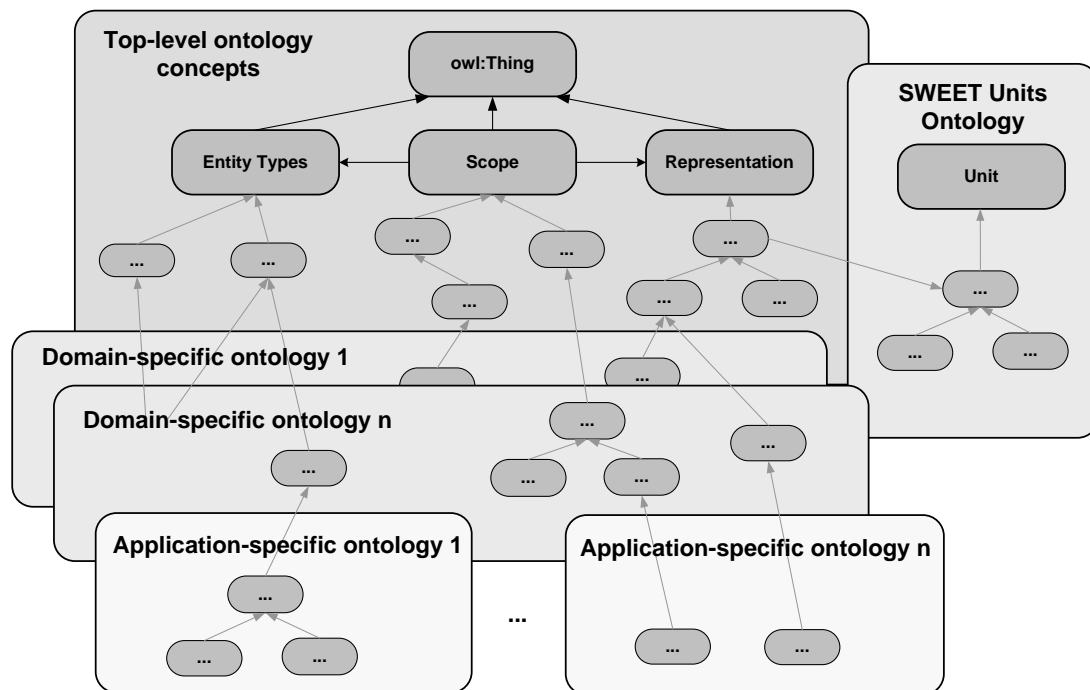
In this chapter we present the generic concepts of our context model. These concepts are implemented in an OWL ontology (see Section 3.3.2). As it is impossible to provide an exhaustive ontology addressing every domain and applicable for every application, we decided to allow a hierarchical composition of the core context ontology, whereby the core ontology is extended by one or more domain or application specific ontologies (e.g. we extended it with an ontology containing all examples presented in this chapter and also used in the demonstrator described in Chapter 11).

For this hierarchical composition, we adopted the concept of the SOCAM project (see Section 4.1.19). The Service-Oriented Context-Aware Middleware (SOCAM) [46] is an architecture for building context-aware services based on a two-level context model. This middleware acquires context information from different sources and interprets it. The context ontology is divided into a two-level hierarchy, distinguishing between common and specific context information. The upper level describes global concepts of

<sup>4</sup>The Unix timestamp is defined as the number of seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970. [http://en.wikipedia.org/wiki/Unix\\_time](http://en.wikipedia.org/wiki/Unix_time). Last visited on Feb 12, 2012.

the ontology and the lower level is divided into several pervasive computing sub-domains, each one of which defines specific details and properties for each scenario. Depending on the situation and the available devices, an appropriate sub-domain is selected from the lower level. When environment changes are detected, the lower level ontology can be dynamically plugged into and unplugged from the upper ontology, hence dynamically changing this association. This mechanism appears to be very reasonable also with respect to resource limited devices. An ontology resulting from the extension of the top-level ontology with a domain-specific ontology can be kept quite small in comparison to a single ontology capturing all potentially involved concepts.

Reichle extended the idea of the SOCAM project by introducing additional hierarchies as depicted in Figure 6.7 [109].



**Figure 6.7:** Hierarchical Composition of the Ontology [109]

Similar to the SOCAM project, the top-level ontology contains the core concepts (classes and properties). This top-level ontology can be extended by various domain-specific ontologies, which in turn can be extended by various application-specific ontologies. In this work, we simply adopt this concept. The concepts presented here are all part of our top-level-ontology. Whereas the concrete examples are part of an extension we made for demonstration issues.

## 6.6 Discussion

The introduced context modelling approach serves as the baseline of the work presented in this document. In the first section of this chapter, we presented the different layers of our context model. The conceptual layer and its concepts have been discussed afterwards



in detail: The core idea of our context model is to specify context information, not only with regard to their information type and the entity it corresponds to, but rather to also specify in which representation the information is provided. We additionally extended this idea not only to the actual information but also to its metadata. The so-called Inter-Representation Operations are used to transform data between the representations. This is also our solution for the requirement “Exchange and interpretation of heterogeneously presented context information” (Requirement 3) described in Section 1.2. Furthermore this is one of the big differences to other existing works. Most of them require a fixed representation for a certain scope, for instance a location is always represented as WGS84 coordinates. As a consequence, Strang et al. introduced the concept of *aspects* and *scales* [134], which has been adapted by Reichle et al. to *scopes* and *representations* [158]. Here, we further extended these concepts in order to also establish arbitrary representations for metadata. This requires an extension of the IRO concept to additionally support the calculation of metadata based on other data.

The evaluation of different context modelling approaches by Strang et al. states that ontology based context models are the most expressive models and fulfil most of their requirements [133]. An often discussed drawback of ontologies is the overhead required to process and reason in an ontology. To tackle this problem, we established the three layers of our context model. Only the first (conceptual) layer contains the ontology. This ontology is used at start time of the application to generate the data structures for handling and storing the context information and its metadata at runtime. The representation of the context information and the according representations of the metadata are used for that purpose. Moreover, a dedicated ontology reasoning component is used during the service discovery, matching and mediation phases to extract the required information how to transform/mediate the context respectively metadata into the requested representation(s). After extracting the required information from the ontology to establish the binding, the actual information exchange between context service and consumer does not require further ontology access respectively reasoning.

Another criticism on ontology based context models is that the handling of heterogeneity by ontologies just pushes the problem into another higher level even though the solution is quite similar to source-code based solutions namely to define standardized data structures to exchange information. However, allowing several representations increases not only the flexibility and re-usability but it is also much easier to adapt applications to the preferences and requirements of different human beings, e.g. with regard to units. As a consequence, our approach directly addresses issues that are also raised by Korpipää et al. [73].



## 7 Context Offer and Query Language

---

*In re mathematica ars proponendi pluris facienda est quam solvendi.  
In mathematics the art of asking questions is more valuable than  
solving problems.*

– Georg Cantor (1845-1918)  
Doctoral thesis (1867)

The previously discussed context model addresses the requirement of “Exchange and interpretation of heterogeneously represented context information” (Requirement 3 in Section 1.2) and builds the baseline of our approach. With the help of the concepts of the context model a common vocabulary is established that allows interpreting the meaning and representation of the exchanged data.

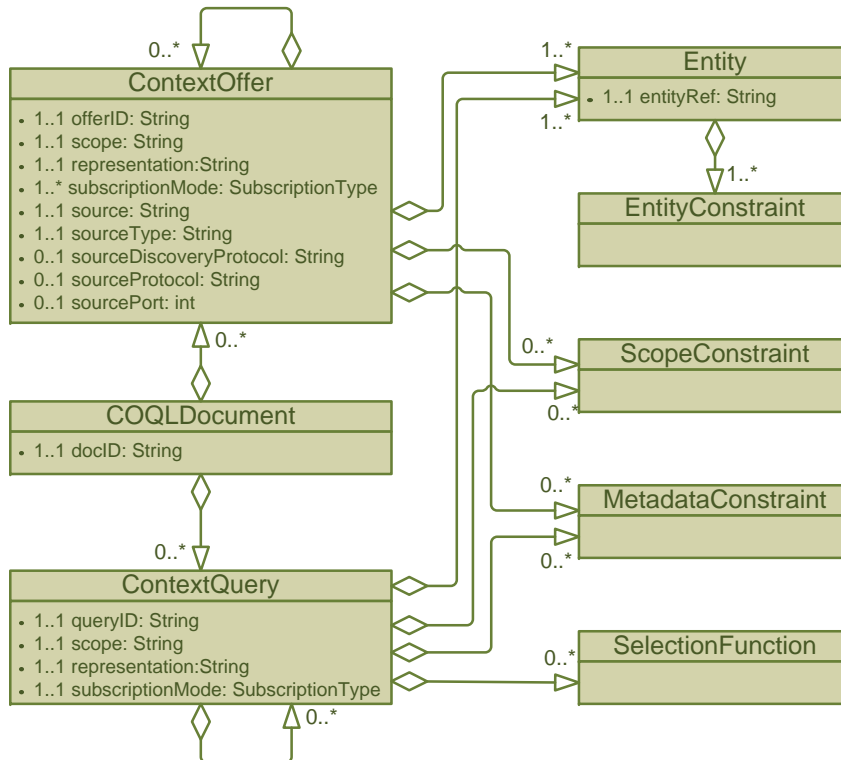
Several context services and consumers can exist in parallel. In order to specify the required and offered context information on the common vocabulary defined by the ontology, a new Context Offering and Query Language (COQL) has been developed.

The COQL provides support for complex filters and conditions in order to precisely define context offers and requests. The corresponding semantic definitions serve as input for the *Discovery and Matching step* described in Chapter 8. The COQL directly addresses the Requirement 4 “Expressing context offers and needs” and are furthermore an important ingredient to solve Requirement 2 “Loose coupling of context providers and consumers” (see Section 1.2 for more details). To allow such a loose coupling, it is necessary that both, the provided and the required context information (including its metadata) are described in detail. Such a description specifies which information (scope) is offered respectively requested, to which entity this information corresponds and in which representations the information is offered respectively requested. Metadata criteria can be expressed to further specify the requested information, e.g. that the freshness of the context information should be smaller than 5 seconds. Optionally, the context consumer defines its preferences regarding cost minimization or quality maximization. These preferences are used in the selection phase to find the ‘best’ context provider for a request.

The COQL is based on the MUSIC Context Query Language (CQL) [159] and the Information Offer and Request Language (IORL) [109]. In contrast to the CQL but like the IORL, the COQL also provides support for context offers. Furthermore, it supports complex filters and conditions like the IORL but in difference to the IORL it also allows different metadata representations and the specification of context selection criteria. The extended support for metadata and selection criteria is also the main contribution of this work compared to its predecessors.

## 7.1 Context Offer and Request

As depicted in Figure 7.1 the central concepts of the COQL are the *COQLDocument*, the *ContextOffer* element, and the *ContextQuery* element<sup>1</sup>. A *COQLDocument* can contain an arbitrary number of context offers and context queries. The *COQLDocument* is used by context services to express both their context needs and offers, e.g. a context reasoning service can require the current position, orientation, and environment volume of the phone to retrieve the current activity of the user.



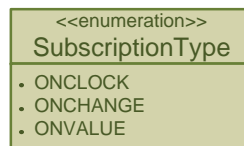
**Figure 7.1:** Overview of the Context Offer and Query Language (COQL)

Both context offer and request reference to the main concepts of the context model: scope, representation, and an arbitrary number of entities. Scope, entities as well as the expected/provided metadata can be further specified with the use of constraints (*EntityConstraints*, *ScopeConstraints* and *MetadataConstraints*). These constraints are explained in detail in Section 7.2. In difference to representation and scope, where the string contains a reference to the corresponding class in the context ontology, the COQL provides an extended support for the entity specification. In the normal case, the string *entityRef* also references to the corresponding *entity type* in the context ontology (e.g. *#Person*). As this entity type can comprise a large number of individuals, it is possible to limit the search space to a specific individual (e.g. the person Roland). This can be done either by referring to the concrete ontology individual or by concatenating the entity

<sup>1</sup>The visualization in Figure 7.1 is adopted from UML (see <http://www.uml.org/>). Classes are depicted as separate boxes and its interconnections by arrows. In this chapter we only use two types of connections: specialization respectively generalization (indicated by ‘is a’) and aggregation, which is indicated by a diamond at the end of the aggregating class.

type, the separator “|” and the name of the individual (e.g. #Person|Roland). This second option is necessary as it cannot be expected that every person is also available as an individual. Consequently, we create implicit new individuals at runtime.

Context offers specify at least one subscription mode. As depicted in Figure 7.2 the subscription mode can be ONCLOCK, ONCHANGE, or ONDEMAND. In the ONCLOCK mode the context service periodically provides context information independent of the fact whether the service can provide new information or not. The frequency as a sensor metadata is specified as metadata constraint. In the ONCHANGE model, the context service informs the context consumer about every change. Both modes require a listener as a context consumer as context information are pushed to the consumer whereas in the ONDEMAND mode the context information are only pulled once from the service.



**Figure 7.2:** COQL: Subscription Modes

In difference to the context queries, the context offers have to provide several additional information on the context service providing the offer: *source*, *sourceType*, *sourceDiscoveryProtocol*, *sourceProtocol*, and *sourcePort*. *Source* (e.g. GPS sensor) and *sourceType* (e.g. context sensor or context reasoner) are used to further describe the context service. These values are only informative and are not used for any decision-making. For this reason, they can contain arbitrary strings. The *sourceDiscoveryProtocol* contains the name of the discovery protocol, which has discovered the context offer, whereas the *sourceProtocol* and *sourcePort* encapsulate the protocol and port required to access the context service. The context queries are additionally associated to optional selection functions. These are described in detail in Section 7.3.

## 7.2 Constraints

In this section, a variety of constraints (*entity constraints*, *scope constraints*, and *metadata constraints*) is described. They are used for further specification of context queries and offers. The different kinds of constraints have in common that they can be either composite or atomic constraints. Composite constraints consist of a logical operator (see Figure 7.3) and of one or more constraints of the same kind. Furthermore, the atomic scope constraints and atomic metadata constraints have in common that they are only unary constraints (e.g. *accuracy*  $\geq 10m$ ). Figure 7.3 illustrates the different operators (*relational*, *string*, and *logical operators*) used in the constraints.

Unlike the atomic scope constraints and the atomic metadata constraints, the atomic entity constraint is a binary constraint as shown in Figure 7.4. The entity constraints are used to further limit the set of entities about which context information are required/offered. The *entityRef* contains in general a reference to an *entity type* of the context ontology. As a large number of individuals can be of that type, the atomic

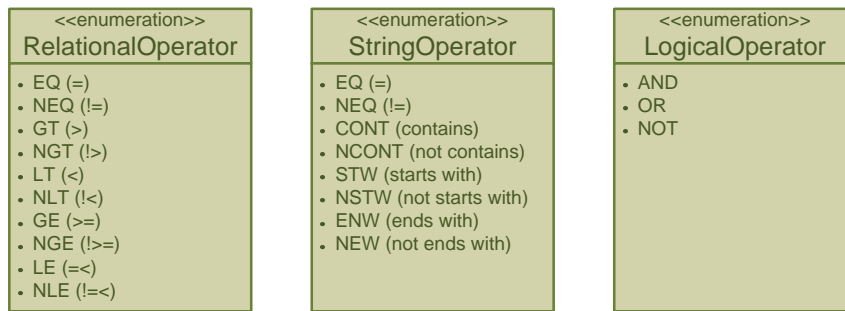


Figure 7.3: COQL: Operators

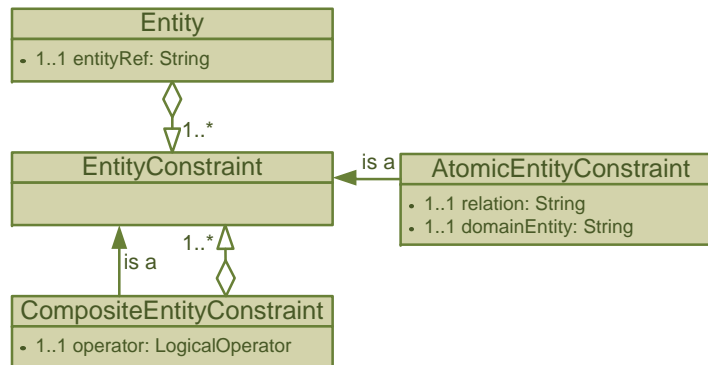


Figure 7.4: COQL: Entity Constraints

entity constraint is used to express *relations*, e.g. ‘work at’, all individuals have to fulfil with regard to a *domain entity*, e.g. ‘University of Kassel’.

Unlike *entity constraints* and *metadata constraints*, which are used for calculating the matching of context offers and queries (see Chapter 8), *scope constraints*, which are depicted in Figure 7.5, are used to filter context information. We distinguish between two kinds of *atomic scope constraints*: *atomic string scope constraints* and *atomic numerical scope constraints*. The only difference is the usage of *string operators* like *contains* in string constraints respectively *relational operators* like  $\geq$  in numerical constraints. Additionally, the *atomic numerical scope constraint* has an additional parameter *delta* to soften the *relational operator*. For example, specifying a delta of 10 within a constraint *accuracy = 100m* results in the constraint *accuracy = 100m ± 10m*.

As shown in Figure 7.6, *atomic metadata constraints* are divided similarly like *scope constraints* into *atomic string metadata constraints* and *atomic numerical metadata constraints*.

### 7.3 Selection Function

Constraints are used to precisely describe context offers and requests. Especially the metadata constraints of context requests are used to reduce the size of the set of context offers that are potentially useful as context provider for the respective request. As this set of potential context providers does not necessarily contain only one offer after

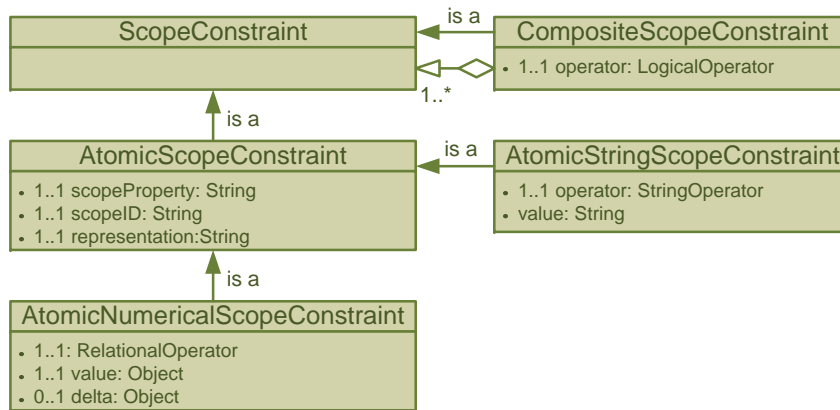


Figure 7.5: COQL: Scope Constraints

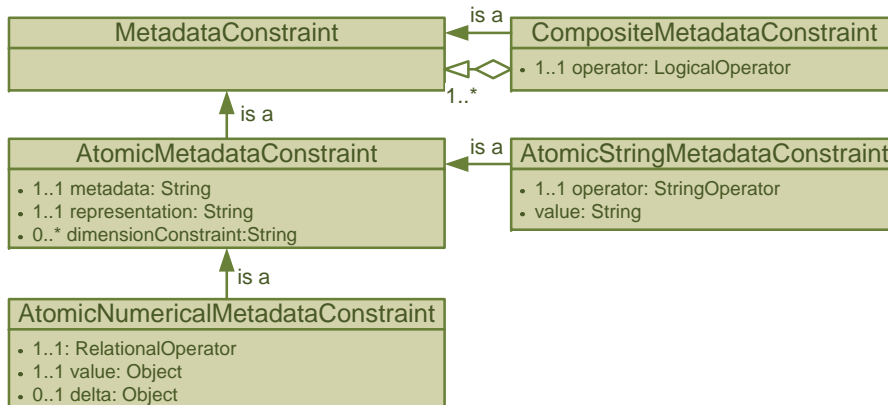


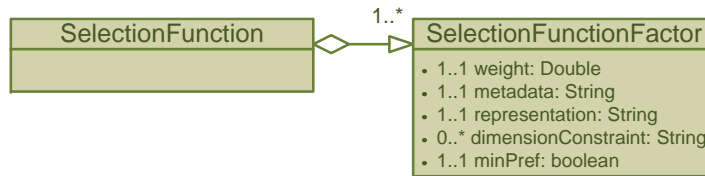
Figure 7.6: COQL: Metadata Constraints

the matching process and as constraints are only optional, a selection function can additionally be specified<sup>2</sup>.

The selection function is an optional part of a context query and is used to select a context offer if more than one context offer is fulfilling the request. While specifying a context request, the developer actually does not know the concrete set of matching context offers and hence he is not able to concretely specify which offer to select. This would also be contradictory to the overall principle of decoupling context offers and requests (see e.g. Requirement 2).

In general the *selection function* used in the selection phase (see Chapter 9) is a utility function (see Section 2.3.2) in form of a weighted sum. In this utility function, the different quality and cost characteristics of a context offer are aggregated to a single number expressing the usefulness of this offer with respect to the user's preferences. The offer from the set of matching context offers with the highest usefulness is selected. In this calculation the user preferences are considered by weighting the quality and cost

<sup>2</sup>If no selection function is specified, all matching offers are rated equally without any user preferences and it is up to the system to select one of the offers. Nevertheless, the selection is always influenced by the system as the system tries constantly to find combined solutions by binding a context provider to more than one query in order to minimize cost.



**Figure 7.7:** COQL: Selection Function

characteristics according to the user preferences.

While specifying a context request, the developer can distribute weights between 0 and 1 to the quality and cost parameters. A weight of 1 expresses the highest importance of the respective parameter. As depicted in Figure 7.7, a *selection function* consists of one or more *selection function factors*. A factor refers to a certain (quality or cost) metadata scope which can be stated more precisely by referencing additionally to one or more dimensions of the representation (by an arbitrary number of *Dimension Constraints*), e.g. instead to weigh the accuracy of a GPS sensor, it is also possible to focus only one the accuracy of the altimeter of this GPS sensor. The Boolean parameter *minPref* expresses the general preference of the user if the minimization or the maximization of the specific metadata are preferred. For example, whereas cost should be minimized, it depends on the quality scope of the metadata and its representation if it is better to minimize or to maximize it.

## 7.4 Example

In this section we introduce a short context query example. For the implementation of the COQL we developed a Java API. The first version of the implementation has been based on Eclipse Modeling Framework (EMF) [131]. EMF is an Eclipse-based modeling framework and code generation facility which provides features like XML serialization and persistence support. Unfortunately it turned out that the generated code is incompatible to the Android platform, which has been selected for the prototypical implementation. For this reason we cleaned up the generated Java code and removed all external dependencies. The current implementation of the COQL as a Java API is also an additional difference to the CQL [159] and the IORL [109] as both languages are XML-based. One drawback of the Java API is its programming language dependency. Nevertheless, the previously described concepts can also be implemented in a language independent way, e.g. in XML. However, we skipped this as it is not essential for the proof of our concepts<sup>3</sup>.

In Listing 7.1 we show an example of a context query expressed with our Java-based implementation:

<sup>3</sup>Within the architecture, a serialization/de-serialization module to transfer a offer respectively request from Java to XML and vice versa is already foreseen. In the architecture described in Chapter 10, the discovery and also the advertisement of context services is done by the discovery service. This service can be extended by so-called discovery plug-ins, which separate concrete implementation and technology dependent aspects for the discovery and advertisement. The previously mentioned serialization/de-serialization module would also only be required within such a discovery module, which does not solely use Java based technology, while for the middleware internal usage the Java based version is used.



```

1 IContextQuery query = coqlFactoryImpl.eINSTANCE.
    createContextQuery ();
2 query.setQueryID("query_position_dummy");
3 query.setRepresentation("#AddressRep");
4 query.setScope("#Position");
5
6 ICOQLEntity entity = coqlFactoryImpl.eINSTANCE.createEntity ();
7 entity.setEntityRef("#User|reichle@vs.uni-kassel.de");
8 query.getEntities().add(entity);
9
10 ISelectionFunction function = coqlFactoryImpl.eINSTANCE.
    createSelectionFunction ();
11 ISelectionFunctionFactor factor1 = coqlFactoryImpl.eINSTANCE.
    createSelectionFunctionFactor ();
12 factor1.setMetadata("#Accuracy");
13 factor1.setRepresentation("#IntegerInMeterRep");
14 factor1.setWeight(0.1);
15 function.getFactors().add(factor1);
16
17 ISelectionFunctionFactor factor2 = coqlFactoryImpl.eINSTANCE.
    createSelectionFunctionFactor ();
18 factor2.setMetadata("#EstimatedSensorCost");
19 factor2.setRepresentation("#IntegerRep");
20 factor2.setWeight(0.9);
21 function.getFactors().add(factor2);
22 query.setSelectionFunction(function);
23
24 IAtomicNumericalMetadataConstraint con1 = coqlFactoryImpl.
    eINSTANCE.createAtomicNumericalMetadataConstraint ();
25 con1.setMetadata("#Freshness");
26 con1.setRepresentation("#DoubleRep");
27 con1.setUnit("#second");
28 con1.setMetadataConstraintID("Freshness");
29 con1.setOperator(RelationalOperator.LT_LITERAL);
30 con1.setValue(Factory.createValue(10));
31 query.getMetadataConstraints().add(con1);

```

**Listing 7.1:** Example of a Context Query

In the example, the current *position* (line 4) as an *address* (line 5) of the user *reichle@vs.uni-kassel.de* is requested. The context provider has to fulfil the *atomic numerical metadata constraint* defined in line 24–31. This constraint is checked during the matching process (see Chapter 8) and is used to filter out inappropriate context providers. Appropriate providers have to provide the information with a freshness of 10 seconds. From the set of appropriate providers, one provider is selected with a selection function. The selection approach is explained in detail in Chapter 9. The information specified in line 10–22 is used to further customize the selection. In this example, the context consumer has the main interest in minimizing the cost and not that much on getting highly accurate data. This is expressed by the weights for the accuracy in line 14 and for the cost in line 20. As the selection function is generally a weighted sum, the utility for a provider  $p$  with regard to the above specified selection function would be calculated with the following formula:

$$u(p) = 0.9 \cdot \frac{cost_p}{max_{\forall provider}(cost)} + 0.1 \cdot \frac{accuracy_p}{max_{\forall provider}(accuracy)} \quad (7.1)$$

In detail, the selection function is a little more complex, but still based on the general idea of the weighted sum. The detailed explanations on the selection approach can be found in Chapter 9.

## 7.5 Discussion

In this chapter we presented our Context Offer and Query Language, which is used to describe the information provided by context services and required by context consumers. By allowing the explicit and detailed specification of both context and requests, it is possible to establish a loose coupling of context consumers and services. This can be compared to service descriptions, e.g. as a WSDL document [143], which are common in service-oriented computing (SOC). However, standard service description approaches are not applicable for our approach, as it requires a lot of modifications to express the information required to describe context offer and request in detail. Similar to SOC, several approaches for the semantic specification of sensors in sensor networks exist. Compton et al. have reviewed twelve of these approaches and came to the conclusion that even if combining several approaches, open issues remain [26]. For example, “questions remain about [...] how to express response model details such as accuracy and how to delineate between and integrate sensors, services and scientific (and other predictive) models” [26].

The COQL is based on the Context Query Language [159] and its successor IORL [109]. Reichle identified that it is not sufficient to only describe a query with a dedicated language [109]. He extended the CQL by support for specification of context offers. But as Reichle considered metadata only with fixed representations, we extended the IORL to also allow different representations for metadata. Additionally, Reichle does not require a selection of context providers. Instead, he fuses the result of matching context offers. As we do not focus on fusion but rather on selection, an appropriate support for expressing the user preferences regarding quality and cost characteristics has been added to the COQL. These preferences are used afterwards in the selection phase (see Chapter 9).

In the future, the COQL should be extended with regard to allow more complex constraints. Currently only unary constraints and their combinations are supported. N-ary constraints would allow to compare variables or also to define complex expressions like  $distanceBetween(friend.position, my.position) < 1km$ , which could be used as e.g. scope constraint. This also requires a notion of ‘distance’ as stated by Becker et al.: “For geometric coordinates, the direct physical distance between two positions can be calculated using well-known formulas like Pythagoras in Cartesian systems. If only symbolic coordinates are modeled then the model must contain explicit definitions of distances between these coordinates, e.g., to define the distance between room number X and the printers in the rooms number Y and Z, since symbolic coordinates do not contain a natural embedment into a metric space. There are other notions of distance that are often more relevant than the direct physical distance. For instance, for a pedestrian it might be impossible to cross a highway. Therefore, a restaurant across the highway with a direct physical distance of 100 m might be farther away than a restaurant with 200 m direct physical distance not located across this highway.” [9, 10].

## 8 Context Offer and Query Matching

*The most exciting phrase to hear in science, the one that heralds new discoveries, is not “Eureka!” (“I found it!”) but rather “hmm . . .that’s funny . . .”*

– Isaac Asimov (1920-1992)  
Unknown source

Using the *Context Offer and Query Language*, context offers and queries are registered to the system. After discovering these offers and queries, the system has to check which context services can provide the requested information and hence satisfy the queries.

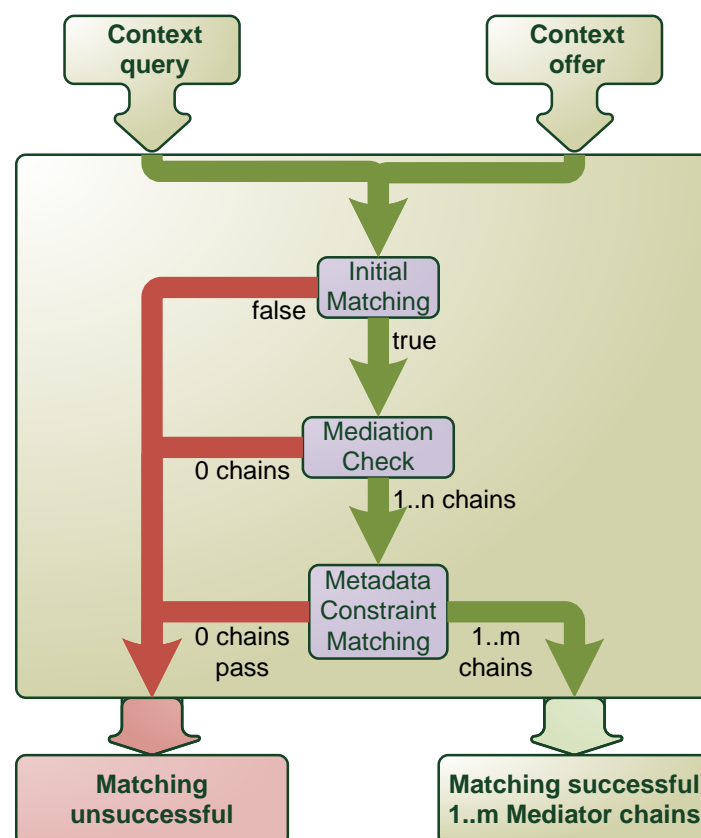


Figure 8.1: Overview on the Context Offer and Query Matching

Figure 8.1 shows the three main steps in the matching process: initial matching, the mediator check, and the metadata constraints matching. During the **initial matching**,

an offer and a query are checked if they fit together with regard to entity, scope and subscription mode. If this is not the case, the matching process stops. Otherwise the **mediation check** is started. While the initial matching focused on scope and entity of the offer respectively the query, the mediation check evaluates if the representations of provided context information and of their metadata fit to the expected representations. If a representation of offered information does not match, the information can be mediated into the requested representation by a chain of mediators. Such a mediator chain consists of one or more Inter-Representation Operations (IROs) and/or Metadata Operators. During this mediation check, it is possible that more than one chain can be created as different IROs respectively chains of IRO can established fulfilling the context request. In this phase not only the mediation of the context data but also of the according metadata is checked. If no mediation is possible and no chains can be created, the matching is unsuccessful. Otherwise, the metadata constraints of the offer and query are checked in the **metadata constraint matching** whether they are satisfiable. As every chain provides a slightly different offer with regard to the metadata constraints (see Section 8.2 for details), it is possible that not all chains, that have been created during the mediation check, pass this phase. But if at least one chain satisfies this test, the matching is successful.

Several approaches exist for matching service requests and service descriptions only on a syntactic level. Other works also exist that extend these syntactical matching with a semantic matching. For example, Bleul et al. “[...]introduce a semantic model and framework for service level brokering where service vendors and consumers can specify their interests in service levels. Service level requests are matched against offers to discover the appropriate service.” [14]. Similarly, SWAPS [100] is a semantic approach for matching WS-A [6] descriptions for automatic partner selection. It uses semantic matching but lacks the ability to transform metrics. This is also the main difference of our concept compared to existing service matching approaches in the domain of ubiquitous computing. Furthermore, we also provide support for mediating the metadata and taking into account the cost caused by the mediation.

The matching process presented in this chapter, contains the complete establishment of mediator chains. A mediator chain converts context information and its metadata from the provided representations to the requested representations. This step has to be part of the matching as

1. without the ability to build such a chain the context service would not be able to satisfy the context request and
2. only after building and using this chain it is possible to reason on the matching of the metadata constraints of offer and request. This is due to the requirement that also the metadata representations have to be transformed (otherwise the constraints would not be comparable) or that the chain influences the metadata (e.g. the accuracy).

## 8.1 Initial Matching

In the *initial matching* phase, context offers and context queries are only coarsely checked. More precisely to pass this phase, the following conditions must hold:

- **Scope matching:** The requested scope is a) the same scope as the offered scope, or b) is a generalization of the offered scope or c) matches with a nested scope<sup>1</sup> of the offer (e.g. the representation of the offer is a composite representation and the requested scope is one of the dimensions of this representation). As mentioned before, the scope constraints are ignored here. They are only used for filtering.
- **Entity matching:** The requested entity or entity type is a) the identical entity respectively entity type as offered or b) is a generalization of the offered entity respectively entity type. Furthermore, the entity constraints must be checked.<sup>2</sup>
- **Subscription mode matching:** The requested subscription mode has to be provided by the context offer.

## 8.2 Mediation Check

After passing the *initial matching*, the representations of the context information and the metadata of both offer and query have to be reviewed. In general, two representations match if

1. the offered representation is the same as the requested representation, or
2. the requested representation is a generalization of the offered representation, or
3. the offered representation can be transformed by one or more IROs into the requested representation.

As explained in Section 6.4, IROs transfer context information of one scope from one representation into another. During this mediation, they can also influence the metadata of this information, e.g. the accuracy. Furthermore, IROs can have additional dependencies on other context information to be able to transfer the actual information. Additionally, metadata operations can be used to derive new metadata from existing metadata.

As depicted in Figure 8.2, various IROs and metadata operations can be linked to a chain to transform the provided information into the requested information. As we allow for different interconnections, we do not directly interconnect IROs but use rather so-called mediators. We distinguish between several kinds of mediators to cope with different kind of required transformation.

Figure 8.2 shows an overview about the general concept of mediator chains containing some of the mediator types, namely the metadata mediator and the IRO mediator. These mediators are used if mediation of context information or metadata is necessary. In difference to the metadata mediator and the IRO mediator, the identity mediator and the extraction mediator serve only as input provider of a mediator chain. Hence, either an identity mediator or an extraction mediator is the first mediator of every mediator chain.<sup>3</sup>

<sup>1</sup>A nested scope is used as a dimension of the representation of the offered scope. For example, a position in WGS84 coordinates contains the nested scopes longitude, latitude, and altitude.

<sup>2</sup>Checking the entity constraints is generally ontology reasoning, which has not been further reviewed in this work as already a lot of existing works target this topic. An overview on these works can be found e.g. in [130].

<sup>3</sup>The chain in Figure 8.2 is missing an identity mediator at the beginning. We removed it in order to improve the readability.

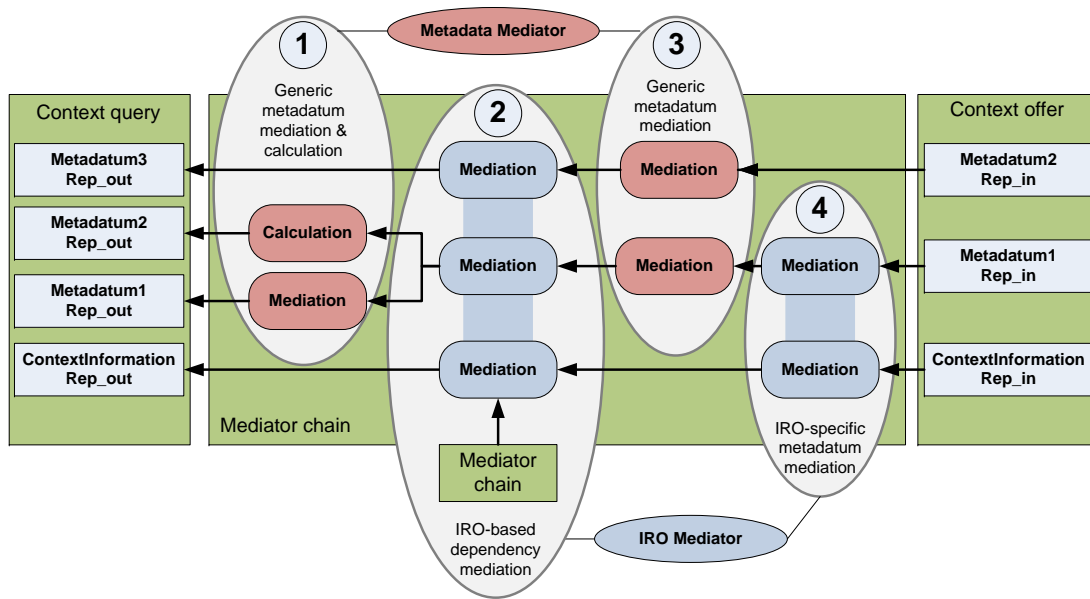


Figure 8.2: Mediator Chain Overview

- **Identity mediator:** The identity mediator is the simplest mediator, as it forwards the input without transforming anything. If a context offer provides exactly the requested information (according to the representations of the actual context information and the associated metadata), the resulting mediator chain only contains an identity mediator. Every successful matching results in at least one mediator chain. This is true, even if this chain contains only one mediator, as this chain is then used to interconnect the loosely coupled context service and consumer.
- **Extraction mediator:** The extraction mediator is used if a requested scope is a nested scope of the offered scope. It extracts the required information from the input and is used as an alternative to the identity mediator serving as starting point of a mediator chain.
- **IRO mediator:** The IRO mediator encapsulates an IRO and can optionally also request additional information such as mediator 2 and 4 depicted in Figure 8.2 by the third mediator which has an additional mediator chain as input.
- **Metadata mediator:** The metadata mediator contains one or more IROs to separately transfer the different types of metadata into the requested representations. Additionally it may contain one or more metadata operations to calculate new metadata (see mediator 1 and 3 in Figure 8.2).

The mediator chain that transfers the offered information and its metadata to the requested information and its metadata is built recursively by Algorithm 8.1. The algorithm is rather straight-forward:

- line 2–4: check if representation of offer and chain match
- line 7–21: walk through the list of IROs and check if
  - line 14–17: requested representation can be provided by an IRO

- line 17–19: requested representation can be provided by one or more chains (recursive call of the method)

The algorithm initially checks if the representations of offer and query are equal (see line 2). If this is the case, the result of the *checkMetadataMediation* algorithm (see Algorithm 8.2) is returned (line 3). In general, the *checkMetadataMediation* method checks whether at the end of a chain a metadata mediation is required in order to fulfil a query or not.

---

**Algorithm 8.1** Algorithm *checkMediation(query, chain<sub>i</sub>, level)* to build mediator chain for input chain *chain<sub>i</sub>* and context query *query*

---

```

1: offer ← chaini.offer
2: if offer.rep = query.rep then
3:   return checkMetadataMediation(query, chaini)
4: end if
5: result ← emptyset
6: iroList ← iroListsForRep.get(offer.rep) ▷ List of IROs transferring data from the
   offer.rep
7: for all i ∈ iroList do
8:   if All dependencies of i are fulfilled then
9:     checkMetadataMediation(generateInputQueryForIRO(i, offer), chaini)
10:    newOffer ← generateOutputForIRO(offer, i)
11:    mediator ← new IROMediator(i, new Offer)
12:    chain ← chaini.copy
13:    chain.append(mediator)
14:    if i.outputRep = query.rep then
15:      checkMetadataMediation(query, chain);
16:      result.add(chain)
17:    else if level ≤ maximalNumberOfHierarchies then
18:      result.addAll(checkMediation(query, chain, level + 1))
19:    end if
20:    if result.size ≥ maximalNumberOfResults then
21:      return result
22:    end if
23:  end if
24: end for
25: return result

```

---

If representation of offer and query are not equal, every IRO that has the same input representation as the representation of the offer is added to a list (line 4). For every IRO in this list, it is first checked if all dependencies are fulfilled (line 8). As explained in Section 6.4.1, an IRO might require additional data in order to transform the data into the required representation. Afterwards it has to be checked if some additional metadata transformations are required before the actual IRO is added. For this proof, the *checkMetadataMediation* method is called (line 9). This algorithm is depicted in Algorithm 8.2 and is explained afterwards in detail. As this method checks whether at the end of a chain a metadata mediation is required in order to fulfil a query or not, it needs

a query and a chain as input parameters. Therefore, we have to generate a new pseudo-query describing the input needs of the IRO. This is done by the *generateInputQueryForIRO* method. This method is also called in line 9 and is further described in Algorithm 8.3.

As now all requirements are fulfilled to perform the IRO, the IRO is now added to the chain (line 13). Before adding the IRO to the chain, a new offer is generated by the *generateOutputForIRO* method first (see Algorithm 8.4) in line 10. Afterwards a new mediator is created in line 11, which is added to a copy of the input chain (lines 12 & 13). The mediator is added to a copy of the chain and not to the origin chain as we are not only searching for one possible mediator chain to transform the requested information into the required representation but rather want to have as many different chains as possible. As every IRO influences the metadata differently, e.g. the accuracy, and the total costs of the chain, the selection process described in Chapter 9 does not only choose the best fitting chain but also selects the best combination of IROs.

Finally a check assures that the last IRO added provides the requested representation (line 14). Then the final metadata mediation is checked (line 15) and the chain is added to the set of results (line 16). If the IRO does not provide the requested representation, the *checkMediation* method is called recursively (line 18) adding its results to the set of chains. The recursive call is interrupted if an IRO provides the requested representation (line 14–17), the maximal number of hierarchy levels is reached (line 17), or no more IROs exist matching the provided representation as input representation. The walk through the set of IROs is aborted, if the maximal number of results (line 20–22) is reached.<sup>4</sup> It must be noted that if a mediator is appended to a mediator chain that provides a representation already provided by another mediator in that chain, an exception is thrown to prevent a mediation cycle or loop in that chain.

---

**Algorithm 8.2** Algorithm *checkMetadataMediation(query, chain)*

---

```

1: offer ← chain.offer.copy
2: metadataMapquery ← generateMetadataMap(query.metadataConstraints)
3: if metadataMapquery.size = 0 then
4:   return chain
5: else
6:   metadataMapoffer ← generateMetadataMap(offer.metadataConstraints)
7:   if metadataMapoffer.size = 0 then
8:     return NULL
9:   else
10:    mediator ← new MetadataMediator
11:    for all dim ∈ metadataMapquery.dimensions do
12:      for all metadataquery ∈ metadataMapquery.get(dim).metadata do
13:        handled ← false
14:        if metadataMapoffer.containsKey(dim) then
15:          metadataoffer ← metadataMapoffer(dim)(metadataquery.scope)

```

---

<sup>4</sup>The parameters *maximalNumberOfHierarchies* and *maximalNumberOfResults* are system parameters, which can be used to tune the number of results returned according to the available system resources. By default both are set to 10, so that maximal 10 chains (each maximally containing 10 IRO mediators) can be constructed. This limit is motivated by the evaluation results. Furthermore, *maximalNumberOfHierarchies* inhibits infinite recursion.



```

16:         if  $metadata_{offer} \neq NULL$  then
17:             if  $metadata_{offer}.rep = metadata_{query}.rep$  then
18:                  $handled \leftarrow true$ 
19:                 continue
20:             else
21:                  $iro \leftarrow findIRO(metadata_{offer}.rep, metadata_{query}.rep)$ 
22:                 if  $iro \neq NULL$  then
23:                      $mediator.add(iro, dim, metadata_{query}.scope)$ 
24:                      $handled \leftarrow true$ 
25:                 else
26:                     return  $NULL$ 
27:                 end if
28:             end if
29:         end if
30:     end if
31:     if  $\neg handled$  then
32:          $ops \leftarrow findOperation(metadata_q.scope, metadata_q.rep)$ 
33:         for all  $op \in ops$  do
34:              $inputAvailable \leftarrow true$ 
35:             for all  $input \in op.inputs$  do
36:                 if  $metadataMap_{offer}.get(dim).contains(input.scope) \wedge$   

 $metadataMap_{offer}.get(dimension).get(input.scope).rep = input.rep$  then
37:                      $inputAvailable \leftarrow inputAvailable \wedge true$ 
38:                 else
39:                      $inputAvailable \leftarrow false$ 
40:                 continue
41:                 end if
42:             end for
43:             if  $inputAvailable = true$  then
44:                  $mediator.add(op, dim, metadata_{query}.scope)$ 
45:             else
46:                 return  $NULL$ 
47:             end if
48:         end for
49:     end if
50: end for
51: end for
52:     if  $\neg(mediator.isEmpty)$  then
53:          $chain.append(mediator)$ 
54:     end if
55:     return  $chain$ 
56: end if
57: end if

```

---

In Algorithm 8.2 the *checkMetadataMediation* method is described which checks whether a metadata mediation is required at the end of a chain so that a query is fulfilled or not:

- line 11–51: check for all metadata requested by the query if

- line 17–20: metadata is provided as requested or
- line 20–28: metadata is provided by a IRO transformation or
- line 33–48: metadata is calculated with a metadata operation

The algorithm starts with the generation of a map containing the different metadata for every dimension by the *generateMetadataMap* method in line 2. If this map is empty for the query, the query does not contain any metadata constraints and does not require any transformation. Hence, the chain is returned (line 3–5). If this map is non-empty but the map for the offer is empty, the mediation is aborted, as obviously the offer does not provide any metadata. Without any metadata, it is not possible to fulfil the request – neither by an IRO nor by any metadata operation as both require metadata as input – (line 7–9).

Now for every dimension (line 11) and the metadata of the query for this dimension (line 12) it is checked whether the offer provides the metadata in the same dimension with the same scope (line 14–16). If this is the case, the algorithm checks – similar to the algorithm above – if an IRO is required to transform the offered metadata into the requested representation (line 17–29). If the offer does not provide the metadata with the same scope as the offer, it is checked if a metadata operation can be used to calculate the requested metadata based on other available metadata. This is done in the second part of the *checkMetadataMediation* method (line 30–56).

To check whether a metadata operation can be used to calculate the requested metadata, first the *findOperation* method is called to find all operations providing information of the required scope and in the required representation (line 32). Next it is checked for all these operations if the input that the respective operation requires, can be provided (line 33–42). If a suitable operation is found, it is also added to the mediator (line 44). Otherwise the mediation is aborted as the required metadata cannot be provided (line 46).

If all dimensions and scopes are checked (line 51) and if the mediator contains at least one IRO or operation, the mediator is append to the chain (line 52–53) and the chain is returned (line 55).

In order to mediate between an offer before an IRO and the requirements of that IRO with respect to the provided metadata and their representations, a query is generated encapsulating these requirements of the IRO and allowing to reuse the previously described method.

Algorithm 8.3 generates this new query based on the provided context offer and the metadata preferences of the provided IRO. As mentioned before, this method is required to check if some additional metadata mediations are required in a chain before an IRO:

- line 1–8: Generate a new query by copying everything except the representation (representation is replaced by the representation of the IRO) and the metadata constraints from the offer.
- line 9–26: Generate new metadata constraints for the new query based on the metadata provided by the offer and the specification of the IRO regarding its influence on metadata (the IRO can influence also the metadata and consequently requests these metadata to have a specific representation).

In detail, *generateInputQueryForIRO* creates a new query with the same entities and scope as the offer (lines 2–4). The representation requested by the query is the input representation of the IRO (line 5). Before returning the new query (line 7), new constraints are generated based on the constraints of the offer by the *generateInputMetadataConstraintsForIRO* function (line 6). This function is depicted in lines 9–26. The function checks for every constraint (line 11) if the IRO requests another representation for the referenced metadata (line 14). In the corresponding constraints the representation is exchanged (lines 15–17). The function is called recursively to transfer also the composite constraints (lines 19–23).

---

**Algorithm 8.3** Algorithm *generateInputQueryForIRO(iro, offer)* to generate a new context query based on the context offer *offer* and the metadata preferences of the IRO *iro* input

---

```

1: function generateInputQueryForIRO(iro, offer)
2:   query ← new Query
3:   query.scope ← offer.scope
4:   query.entities ← offer.entities
5:   query.rep ← iro.inputRep
6:   query.metadataConstraints ← generateInputMetadataConstraintsForIRO
   (offer.metadataConstraints, iro)
7:   return query
8: end function
9: function generateInputMetadataConstraintsForIRO(inputMetadataConstraints,iro)
10:  result = new List < MetadataConstraint >
11:  for all constraint ∈ inputMetadataConstraints do
12:    temp ← constraint
13:    if constraint is AtomicNumericalMetadataConstraint ∨
   constraint is AtomicStringMetadataConstraint then
14:      newRep ← iro.getInputMetadataRepresentationForScopeAndDim...
   (constraint.getMetadata(), constraint.getDimensionConstraint);
15:      if newRep ≠ null then
16:        temp.rep ← newRep
17:      end if
18:      result.add(temp)
19:    else if constraint is CompositeMetadataConstraint then
20:      temp.constraints.clear
21:      temp.constraints ← generateInputMetadataConstraintsForIRO
   (constraint.constraints)
22:      result.add(temp)
23:    end if
24:  end for
25:  return result
26: end function

```

---

For example, a context offer provides the current position of a user as WGS84 coordinates and additionally the accuracy of this position in meter. As the context query requests the position as an address, an IRO can be used to mediate from the WGS84 coordinate

to an address. With this mediation also the accuracy is influenced. However, the IRO, which asks for the context information in WGS84 coordinates as input data, expects the accuracy to be represented in miles. To express this requirement, an according input query for that IRO is generated to check the required metadata mediation between offer and IRO.

---

**Algorithm 8.4** Algorithm *generateOutputForIRO(iro, offer)* to generate a new context offer based on the context offer *offer* and the metadata preferences of the IRO *iro* output

---

```

1: function generateOutputForIRO(iro, offer)
2:   newOffer  $\leftarrow$  new Offer
3:   newOffer.scope  $\leftarrow$  offer.scope
4:   newOffer.entities  $\leftarrow$  offer.entities
5:   newOffer.rep  $\leftarrow$  iro.outputRep
6:   newOffer.metadataConstraints  $\leftarrow$  generateOutputMetadataConForIRO
   (offer.metadataConstraints, iro)
7:   return newOffer
8: end function
9: function generateOutputMetadataConstraintsForIRO(inputMetadataConstraints,
   iro)
10:  result = new List < MetadataConstraint >
11:  for all constraint  $\in$  inputMetadataConstraints do
12:    temp  $\leftarrow$  constraint
13:    if constraint is AtomicNumericalMetadataConstraint  $\vee$ 
   constraint is AtomicStringMetadataConstraint then
14:      newRep  $\leftarrow$  iro.getInputMetadataRep...ForScopeAndDimension
   (constraint.getMetadata(), constraint.getDimensionConstraint);
15:      if newRep  $\neq$  null then
16:        temp.rep  $\leftarrow$  newRep
17:        temp.value  $\leftarrow$  iro.performWorstCaseMetadataConversion
   (constraint.value)
18:      end if
19:      result.add(temp)
20:    else if constraint is CompositeMetadataConstraint then
21:      temp.constraints.clear
22:      temp.constraints  $\leftarrow$  generateInputMetadataConstraintsForIRO
   (constraint.constraints)
23:      result.add(temp)
24:    end if
25:  end for
26:  return result
27: end function

```

---

The next operation called in the main Algorithm 8.1 is the *generateOutputForIRO* method, which is described in detail in Algorithm 8.4. This algorithm calculates a new offer which is provided by a chain after executing the last IRO in that chain. In general, this function is comparable with the Algorithm 8.3. The main difference is that also the values of some

metadata constraints have to be adjusted (line 17). For this purpose, every IRO has to provide a *performWorstCaseMetadataConversion* method to calculate a new value of the constraint. Similar to Algorithm 8.3, the function to transform the metadata constraints is called recursively to transfer also the composite metadata constraints.

### 8.3 Metadata Constraint Matching

The final phase of the complete matching process is to check whether the metadata constraints of context offer and query fit or not.<sup>5</sup> This is only possible after the mediation check as the mediation process may influence the metadata and as a consequence the constraints, which in general express the upper, lower limit, or average value of certain metadata. A metadata constraint  $c_o$  of an offer  $o$  and a constraint  $c_q$  of a query  $q$  match only if and only if:

1. **Scope matching:** The scope of the offer metadata constraint  $c_o$  equals the scope of the request metadata constraint  $c_q$  or the scope of  $c_q$  is a generalization of the scope of the offer constraint  $c_o$ .
2. **Representation matching:**
  - a) The representation of the offer metadata constraint  $c_o$  equals the representation of the request metadata constraint  $c_q$
  - b) or the representation of  $c_q$  is a generalization of the representation of the offer constraint  $c_o$ ,
  - c) or the metadata of the offer can be transformed into the representation of the request constraint using an IRO.
3. **Constraint satisfaction:** The conjunction of both constraints  $c_o \wedge c_q$  has to be satisfiable.

The first two requirements are already checked within the previous mediation steps. Hence, we can assume in this phase that two or more constraints of either offer or query with the same scope also have the same representation and are comparable.

In general, constraint programming is a programming paradigm wherein relations between variables are stated in the form of constraints allowing users to describe problems in a declarative way. A possible solution has to satisfy all these constraints. In our approach, context provider and context consumer can use constraints to detail the description of provided respectively requested context information. As stated by Hofstedt et al. [57], constraint logic programming is based on first-order logic as constraints are nothing else than first-order expressions. Whereas a constraint solver provides concrete variable assignments which fulfil the set of constraints, this is not required in our approach. Instead we want to know if the constraints are satisfiable in general.

Unfortunately, the satisfiability problem for first-order logic in general is undecidable [23]. But Löwenheim proved that of first-order logic in which all predicate letters are monadic and which does not contain function letters is decidable [83]. Fortunately, it

---

<sup>5</sup>The topic of this section has also been subject of the Bachelor Thesis of Alexander Kohout [69], which has been supervised by the author of this thesis.

is possible to transform the metadata constraint into this so-called *monadic predicate calculus* as follows:

- **Atomic constraints:** The relational or string operator (see Figure 7.3) of the constraint is modelled as a predicate symbol of arity 2 and the two parameters of this predicate are the scope of the constraint and the value of the constraint. This predicate is only monadic, as only the scope is a free variable, whereas the value is bound.
- **Composite constraints:** The formulae for the 1..n constraints contained in the composite constraint are associated with  $\wedge$  (if logical operator is AND),  $\vee$  (logical operator is OR) or  $\neg$  (logical operator is NOT).
- **Top-level list of constraints:** As described in Section 7.1, a context offer or request can contain an arbitrary number of metadata constraints, which have all to be satisfied in order to satisfy the offer or request. For this reason, the formulae for these constraints (either atomic or composite) are associated with  $\wedge$  (colloquial *and*).

Following these transformation rules, the metadata constraints of the query shown in Listing 8.1 result in the following formula: <sup>6</sup>

$$\begin{aligned} & (LessThan(\#Freshness, 10) \wedge (LessThan(\#Accuracy, 100) \\ & \vee GreaterThan(\#Reliability, 75))) \end{aligned} \quad (8.1)$$

```

1 IContextQuery query = coqlFactoryImpl.eINSTANCE.
  createContextQuery();
2 [...]
3 IAtomicNumericalMetadataConstraint con1 = coqlFactoryImpl.
  eINSTANCE.createAtomicNumericalMetadataConstraint();
4 con1.setMetadata("#Freshness");
5 con1.setRepresentation("#DoubleRep");
6 con1.setUnit("#second");
7 con1.setMetadataConstraintID("Freshness");
8 con1.setOperator(RelationalOperator.LT_LITERAL);
9 con1.setValue(Factory.createValue(10));
10 query.getMetadataConstraints().add(con1);
11 IAtomicNumericalMetadataConstraint con2 = coqlFactoryImpl.
  eINSTANCE.createAtomicNumericalMetadataConstraint();
12 con2.setMetadata("#Accuracy");
13 con2.setRepresentation("#DoubleRep");
14 con2.setUnit("#meter");
15 con2.setMetadataConstraintID("Accuracy");
16 con2.setOperator(RelationalOperator.LT_LITERAL);
17 con2.setValue(Factory.createValue(100));
18 IAtomicNumericalMetadataConstraint con3 = coqlFactoryImpl.
  eINSTANCE.createAtomicNumericalMetadataConstraint();
19 con3.setMetadata("#Reliability");
20 con3.setRepresentation("#DoubleRep");

```

<sup>6</sup>As described above, we can assume that constraints of the same scope also have the same representations/units. Hence, we can ignore this information when transforming the constraints into a formula.

```

21 con3.setUnit("#percent");
22 con3.setMetadataConstraintID("Reliability");
23 con3.setOperator(RelationalOperator.GT_LITERAL);
24 con3.setValue(Factory.createValue(75));
25 ICompositeMetadataConstraint con4 = coq1FactoryImpl.eINSTANCE.
    createCompositeMetadataConstraint();
26 con4.setOperator(LogicalOperator.OR_LITERAL);
27 con4.getConstraints().add(con2);
28 con4.getConstraints().add(con3);
29 query.getMetadataConstraints().add(con4);

```

**Listing 8.1:** Example of Metadata Constraints

In order to prove the satisfiability of the constraints of both offer and query, the formulae for the constraints are conjuncted ( $\wedge$ ). The satisfiability of this conjunction can be proven by any method presented in Section 3.2. In our prototypical implementation, we use the *analytic tableaux method* (see Section 3.2.2). As our formulae do not contain any quantifiers ( $\forall$  or  $\exists$ ), we can use the simple method for propositional logic (see Section 3.1.1).

Following Definition 3.4 and Theorem 3.1 the conjunction of the formulae is satisfiable, if the resulting tableaux is completed and open. The analytic tableaux method has several advantages over e.g. the resolution method (see Section 3.1.2 and Section 3.2.1). For instance the method of analytic tableaux does not require an input formula to be in a certain normal form. Another important advantage of tableaux method compared to resolution method is that the tableaux method is able to stop analysing a formula without reducing the formula to its atomic components.

We construct the analytic tableau as defined in Definition 3.3 but change the rule when to close a branch. For propositional logic a branch is closed if it contains both a formula  $F$  and its negation  $\neg F$ . In difference to this rule, we support two different modes to close a branch: *weak* and *strong consistency check*.

Before defining these consistence classes, we have to introduce the domain of metadata of a certain scope (e.g. freshness).

**Definition 8.1** (metadata domain). The domain of a metadata scope is the complete set of possible values of metadata of that scope. Based on a metadata constraint, the domain  $D_A$  of the corresponding metadata scope  $A$  is defined in Table 8.1.

Based on this definition, we can define weak and strong consistency as follows:

**Definition 8.2** (weak consistency). A constraint  $A$  is *weakly consistent* to a constraint  $B$  if the intersection of both domains is non-empty, thus if  $D_A \cap D_B \neq \emptyset$ .

**Definition 8.3** (strong consistency). A constraint  $A$  is *strongly consistent* to a constraint  $B$  if the domain of  $A$  is a subset of the domain of  $B$ , thus if  $D_A \subset D_B$ .

Based on the previous definition, constraints are satisfiable a) for some of the possible variable assignments if the constraints are *weakly consistent* and b) for all possible variable assignments if the constraints are *strongly consistent*. From the definition it is clear that strong consistency implies weak consistency as from  $D_A \subset D_B$  follows  $D_A \cap D_B = D_A \neq \emptyset$ .

Operator	Short name	Predicate	Metadata domain $D_A$
Equal	EQ	$EQ(A, x)$	$D_A = \{x\}$
Not equal	NEQ	$NEQ(A, x)$	$D_A = \{y \in \mathcal{R}   y \neq x\}$
Greater than	GT	$GT(A, x)$	$D_A = \{y \in \mathcal{R}   y > x\} = (x, \infty)$
Not less or equal	NLE	$NLE(A, x)$	
Not greater than	NGT	$NGT(A, x)$	$D_A = \{y \in \mathcal{R}   y \leq x\} = (-\infty, x]$
Less or equal	LE	$LE(A, x)$	
Less than	LT	$LT(A, x)$	$D_A = \{y \in \mathcal{R}   y < x\} = (-\infty, x)$
Not greater or equal	NGE	$NGE(A, x)$	
Not less than	NLT	$NLT(A, x)$	$D_A = \{y \in \mathcal{R}   y \geq x\} = [x, \infty)$
Greater or equal	GE	$GE(A, x)$	

**Table 8.1:** Definition of the Metadata Domain

To check whether a branch of the analytic tableaux has to be closed or not, we check if all atomic constraints of the offer are weakly or strongly consistent<sup>7</sup> to the corresponding atomic constraints of the query with the same scope. If we detect inconsistency, the branch has to be closed. Per definition, weak consistence allows that some of the provided context information do not fulfil the request<sup>8</sup> (more precisely the metadata constraints of the request), whereas all context information fulfil the request if we check for strong consistency. Weak consistency has the advantage that the repertory of context offers to fulfil a query is larger than the repertory with strong consistency. As a consequence, the probability of not finding an offer fulfilling the request is smaller than with strong consistency.

To explain the complete metadata constraint matching, we first introduce a new context offer:

```

1 IContextOffer offer = coqlFactoryImpl.eINSTANCE.
   createContextOffer ();
2 [...]
3 IAtomicNumericalMetadataConstraint con1 = coqlFactoryImpl.
   eINSTANCE.createAtomicNumericalMetadataConstraint ();
4 con1.setMetadata("#Freshness");
5 con1.setRepresentation("#DoubleRep");
6 con1.setUnit("#second");
7 con1.setMetadataConstraintID("Freshness");
8 con1.setOperator(RelationalOperator.LT_LITERAL);
9 con1.setValue(Factory.createValue(5));
10 offer.getMetadataConstraints().add(con1);
11 IAtomicNumericalMetadataConstraint con2 = coqlFactoryImpl.
   eINSTANCE.createAtomicNumericalMetadataConstraint ();
12 con2.setMetadata("#Accuracy");
13 con2.setRepresentation("#DoubleRep");

```

<sup>7</sup>By default, we check for weak consistency. The middleware can be configured by a system property to check for strong consistency.

<sup>8</sup>This non-fitting context information is filtered out by the middleware before transferring them to the context consumer.



```

14 con2.setUnit("#meter");
15 con2.setMetadataConstraintID("Accuracy");
16 con2.setOperator(RelationalOperator.GT_LITERAL);
17 con2.setValue(Factory.createValue(50));
18 offer.getMetadataConstraints().add(con2);

```

**Listing 8.2:** Example of Metadata Constraints 2

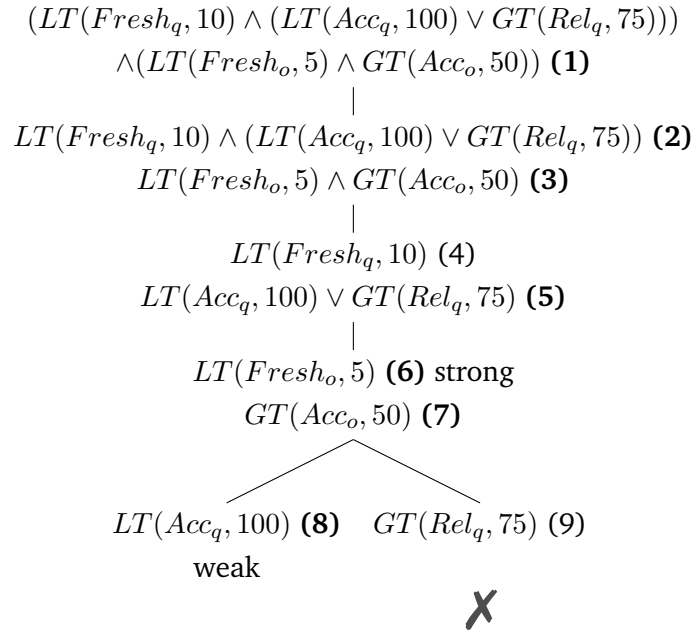
The metadata constraints of the query shown in Listing 8.2 result in the following formula:

$$(LT(\#Freshness, 5) \wedge GT(\#Accuracy, 50)) \quad (8.2)$$

In order to proof satisfiability of the constraints of the query in Listing 8.1 and the offer in Listing 8.2, we have to construct the analytic tableaux for the conjunction of the formulae Equation 8.1 and Equation 8.2, hence for the following formula:<sup>9</sup>

$$(LT(Fresh_q, 10) \wedge (LT(Acc_q, 100) \vee GT(Rel_q, 75))) \wedge (LT(Fresh_o, 5) \wedge GT(Acc_o, 50)) \quad (8.3)$$

The resulting tableaux is depicted in Figure 8.3.



**Figure 8.3:** Example for Metadata Constraint Matching

The tableaux depicted in Figure 8.3 was constructed as defined in Definition 3.3. The conjunction of the formulae for offer and query metadata constraints are placed in the origin (1). Now following the first  $\alpha$  rule  $\frac{X \wedge Y}{X \quad Y}$  we have to adjoin  $X$  and  $Y$  to the

<sup>9</sup>To improve the readability, we abbreviated the parameter names: Fresh = #Freshness, Acc = #Accuracy, Rel = #Reliability. Furthermore, we added indices  $o$  and  $q$  to indicate whether a constraint is associated to the offer or the query, respectively.

tableaux:  $LT(Fresh_q, 10) \wedge (LT(Acc_q, 100) \vee GT(Rel_q, 75))$  (2) and  $LT(Fresh_o, 5) \wedge GT(Acc_o, 50)$  (3). The first  $\alpha$  rule is used again for (2) which results in  $LT(Fresh_q, 10)$  (4) and  $LT(Acc_q, 100) \vee GT(Rel_q, 75)$  (5). Afterwards the rule is used for (3) resulting in  $LT(Fresh_o, 5)$  (6) and  $GT(Acc_o, 50)$  (7). When adding (6), we can perform the first consistency check for the tree. As it already contains an atomic constraint with the same scope (here Freshness), we can perform the check for  $LT(Fresh_q, 10)$  in (4) and  $LT(Fresh_o, 5)$  in (6). Based on Definition 8.1 and Definition 8.3, we detect strong consistency and continue.  $GT(Acc_o, 50)$  in line (7) can be checked so far, as another atomic constraint with the same scope is inside the tableaux.

Following the second  $\beta$  rule  $\frac{X \vee Y}{X \mid Y}$ , (5) branches into  $LT(Acc_q, 100)$  in (8) and  $GT(Rel_q, 75)$  in (9). Now we can also check the consistency for (8), which results in weak consistency. (9) cannot be checked as the tableaux is now completely expanded. But as all atomic formulae in the first branch are weakly consistent, the tableaux is still open. As a consequence, the query and offer constraints match.

## 8.4 Example

This section illustrates the matching process by an example. As shown in Figure 8.4 there is one context query, three context offers, three IROs, and one metadata operation registered at the system. The clouds in the offers and queries indicate metadata constraints, whereas the clouds in the IROs describe the influence on the metadata by the according IRO. Several clouds for an offer are interpreted like in the COQL as a conjunction of these constraints.

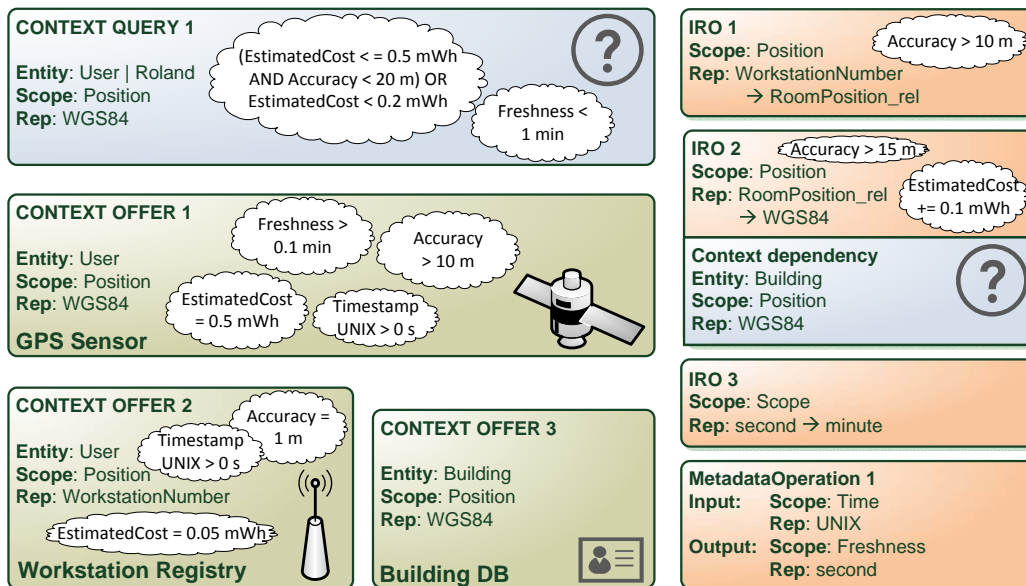


Figure 8.4: Example for the Context Offer and Query Matching

The context query requests the *position* of the *user* with the ID *Roland* represented in *WGS84* coordinates. The context consumer is willing to pay a higher price (here expressed in energy up to 0.5 mWh) if the according offer can provide the requested position with



The second offer provides a *position* of a *user* in form of a *workstation number*. Furthermore, it provides metadata like timestamp, accuracy, and estimated cost. Similar to the first offer, the initial matching is quickly passed as entity and scope are similar to the requested ones. However, the mediator check is much more complex due to the non-fitting representations. The result of this phase is a mediator chain like shown in Figure 8.6.

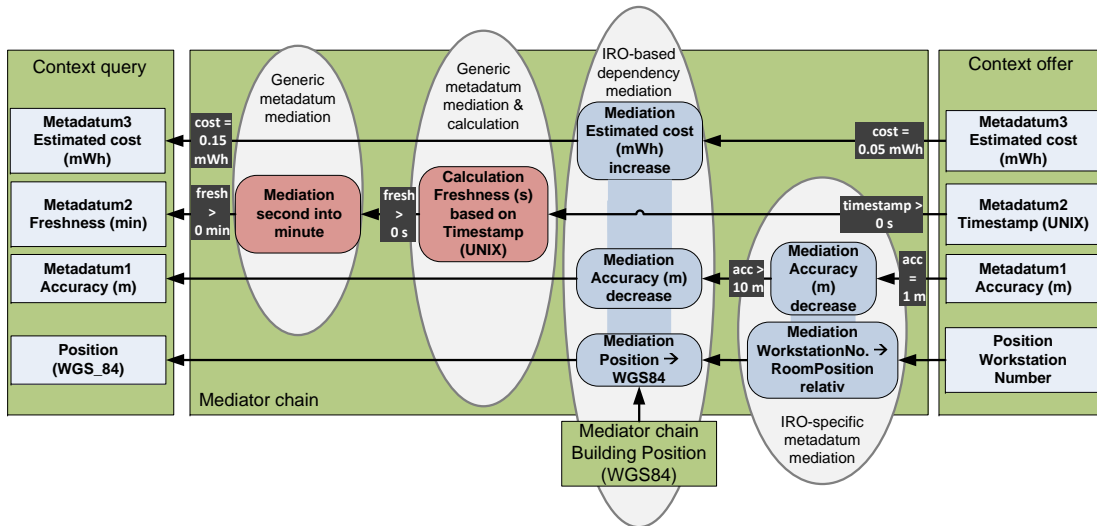


Figure 8.6: Mediator Chain Example

The *checkMediation* algorithm (Algorithm 8.1) initially appends the *IRO 1*, as it is the only available IRO with the representation *WorkstationNumber* as input representation. This IRO transfers the position into the relative room position (*RoomPosition\_rel*) and decreases the accuracy from 1 meter (initial offer) to minimum of 10 meters. Before appending the IRO to the chain, the algorithm checks if a metadata mediation is required to bring the influenced metadata into the representation requested by the IRO. This is not the case in this example. As the provided representation of *IRO 1* is not the requested representation, the algorithm appends the next IRO (*IRO 2*). Whereas the first IRO only influences the accuracy, the second IRO influences two types of metadata (cost and accuracy) and has an additional context dependency. Again the algorithm checks if a metadata mediation is required before the IRO. Due to the additional context dependency, the algorithm also checks if an appropriate offer is available before proceeding. This is the case in our example, as *Offer 3* provides the requested information: the position of the building in WGS84 coordinates. With the help of this information, the IRO is able to transform the information provided by the previous IRO to the requested representation (WGS84).

As the chain offers the information in the requested representation, the *checkMediation* algorithm stops after calling the *checkMetadataMediation* algorithm. While running this algorithm, it turns out that the offer cannot provide metadata with the scope *Freshness*. Therefore, a new mediator is appended to the chain containing the metadata operator *MetadataOperation1* which calculates the metadata scope *Freshness* based on the *Unix timestamp*. As this operation cannot really provide any detailed constraints regarding the provided metadata, a simple metadata constraint is added to the offer



not only matching of offer and query is checked, but additionally the mediator chains are created. These chains interconnect the loosely coupled context consumer and context service. Within such a chain, a couple of mediators (at least one) are called sequentially to transform the data and its associated metadata into the requested representation. Setting up these chains is already important in this phase: If it is not possible to create the chains, offer and query do not match and without the creation of the chain it is not possible to check the metadata constraint satisfaction. After the creation of mediator chains, the constraints expressed in a context offer (the adopted offer of the chain and not the original offer) and the constraints detailing a context query are comparable.

The autonomous establishment of the mediator chains has to be highlighted. Other approaches, like CoCo by Buchholz et al., can also compose and transform different context information in order to retrieve the requested information in the required representation, but here the context consumer has to provide a model containing the description of the mediator chain.

To estimate the complexity of Algorithm 8.1 *checkMediation*, the complexity of the other algorithms has to be determined first:

- The algorithm *checkMetadataMediation* described in Algorithm 8.2 in worst case has to iterate through the list of metadata used in the constraints (cardinality is  $\#meta$ ) and through the list of all metadata operators (cardinality is  $\#op$ ). Hence, its worst case complexity is  $\mathcal{O}(\#meta + \#op) = \mathcal{O}(n)$  and hence linear.
- Algorithm 8.3 *generateInputQueryForIRO* and Algorithm 8.4 *generateOutputForIRO* iterate through the list of metadata constraints (cardinality is  $\#con$ ) of the offer provided as an input parameter. For that reason, the complexity of both algorithms is  $\mathcal{O}(\#con) = \mathcal{O}(n)$  and thus also linear.

Based on these estimations, the complexity of *checkMetadataMediation* can be derived. In the worst case, the algorithm is called  $\#maximalNumberOfHierarchies$  times recursively as no IRO provides the requested representation. In every call,  $\#maximalNumberOfResults$  IROs are used to generate new variants of the initial mediator chain. Furthermore, during every call *checkMetadataMediation* and *generateInputQueryForIRO* are called once. For this reason, the complexity can be calculated as  $\#maximalNumberOfHierarchies \cdot \#maximalNumberOfResults \cdot (\#meta + \#op + \#con)$ .

During the metadata constraint matching we reuse a standard approach (analytic tableaux as introduced by Smullyan [128]). The complexity of this approach has been discussed by e.g. Massacci [91] and estimated to  $\mathcal{O}(2^{n^2})$ , whereas in our case  $n = \#offer.constraints + \#query.constraints$ . This is only the worst case complexity. The constraint matching performs in average and best case much better. This will be discussed in Chapter 12 in detail.

Regarding the consistency check during the constraint matching, the strong consistency is preferable as it guarantees that context data meet the requirements of the context consumer and do not require any additional filtering. Nevertheless, it might be challenging to find offers with strong consistence. It is future work to integrate the result of the consistence check in the actual selection and not in the matching phase in order to handle this problem dynamically. A context offer with strong consistence could be rated better than an offer with weak consistence.

At this point, we will discuss a weakness of our current approach: In Algorithm 8.4 the method *performWorstCaseMetadataConversion* is called in order to revise the metadata constraints according to the influence of the IRO regarding the metadata. This method has to be provided by every IRO and is part of its implementation. But it is obvious that the developer at least requires methodological support to estimate the influence on metadata caused by an IRO conversion. This support is also required to estimate respectively calculated the QoC and CoC at all, which is not in the focus of this thesis. We rather focus on providing a generic frame which can be used with both highly accurate and measured QoC and CoC or with only roughly estimated values for QoC and CoC. This generic approach is our main contribution in this chapter and with this approach we also differ from existing approaches, which focus on the transformation of context information but neglecting the influence on the metadata.





## 9 Context Service Selection

---

*There is a theory which states that if ever anybody discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened.*

– Douglas Adams (1952-2001)  
The Hitchhiker's Guide to the Galaxy (1979)

In the previous chapter, the matching approach checks if a binding of context provider and context consumer via the establishment of one or more mediator chains is possible or not. By any means, it is possible that several context services exist for which such a chain can be established. Additionally, it is possible that a context provider can serve as input provider for more than one mediator chain and thus for more than one request.

To meet this fact, we are not searching for a single offer respectively mediator chain for each context request. We are rather searching for an optional set of context offers respectively chains to fulfil all requests system-wide. By a selection, the total cost should be minimized while the requirements of the different requests are fulfilled as good as possible. Consequently, the selection task is a multi-objective optimization (MOO) problem [25, 94]. In an MOO, two or more conflicting objectives are optimized to certain constraints. In our case we have to find an optimal set of context providers for one query with, for example, costs as little as possible, whereas a second request is only interested in maximizing the accuracy. In most cases increasing the accuracy also means increasing the resulting cost. As a result, an optimal solution for one of the requests is in conflict with the other request. Hence, the best solution for both requests is a so-called *Pareto Optimum* [25], meaning that it is not possible to select another set of context providers respectively mediator chains without hurting at least one request.

One of the general challenges of this work is the dynamic activation and deactivation of required or not required context consumers (Requirement 5). This leads to the problem that no up-to-date and highly accurate description of the provided properties of the context consumers can be assumed but rather the metadata constraints in the context offer describe the worst or at best the average case. To estimate the worst and/or average values of the metadata, historical values can be taken into account. In this estimation the very dynamic nature of context data has to be taken into account: Context data and its metadata can change very often. As a consequence the age of historical context data has to be considered when adjusting worst or average values (see Section 9.3). Last but not least, the selection should favour solutions that minimize the total cost, e.g. by supporting the sharing of context offers as input provider (see Requirement 7).

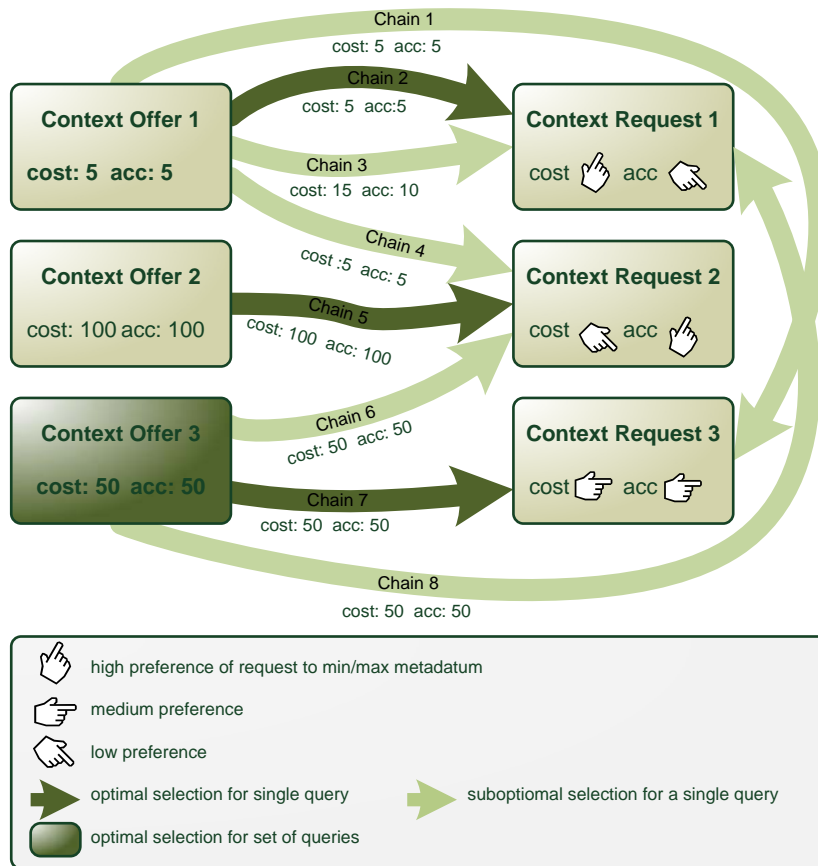
Typically, service selection problems in Service-Oriented Computing (SOC) are limited to find one service for one service request. This is a fundamental difference to our MOO problem. In detail, the focus of service selection in SOC is on finding one service instance for every variation point or abstract service in a service composition which holds the specific constraints and thereby, e.g. optimizes some objective function. Typically the selection does not care about sharing services, as these are generally stateless and only monetary cost are brought to account. In contrast, in our approach also costs (e.g. resource usage or energy consumption) are taken into account.

In order to achieve this, our approach evaluates a set of context offers with respects to the preferences of a set of context consumers. Services in general (like web services) can be assumed to be rather static with regard to changes of their metadata. Even if there are changes it is possible to adjust the service description used to select an optimal service. Services can be assumed to run permanently and can be monitored continuously in contrast to our context services, which are often deactivated. In the SOC domain, service mediation is another important topic [54, 93]. However many approaches for service mediation rather focus on the transportation/exchange layer and only few really target message conversion. Most approaches rely furthermore on manual configuration of the mediation where mediators act like proxies hiding the underlying service. This is why a comprehensive process for automatically establishing the mediation, as described in the previous chapter, is not required. To the best of our knowledge, none of the approaches in SOC consider mediation and selection together but rather focus on both issues separately. The drawback of this approach is that it is not possible to use a service as a source for several mediation chains and to calculate the total cost for the actual cost and the separate mediation processes.

In this chapter, we present our selection approach which provides an optimal solution for a set of context consumers, and minimizes the total costs while satisfying the consumers requests as good as possible. Context services can be shared to serve as input provider for several mediator chains in parallel, which reduces resource consumption. Additionally, the selection process estimates the worst or average metadata values used based on historical context data.

## 9.1 Motivating Example

In this section we motivate the problems through using an example. Figure 9.1 shows three context offers and requests and seven mediator chains between the offers and the requests. For instance, *Context Offer 1* serves as input provider for four mediator chains (*Chain 1 – 4*). Every context offer expresses its cost, e.g. 5 for *Context Offer 1*, and the degree of accuracy, e.g. 5 for *Context Offer 1*. Every chain contains the resulting cost and accuracy, e.g. *Chain 3* increases the cost to 15 whereas it also increases the accuracy to 10. Mediator chains only containing the Identity Mediator (see Section 8.2) and hence merely establishing the bindings, have the same metadata conditions as the input provider, e.g. *Chain 1* has the same cost and accuracy as its input provider *Context Offer 1*. In contrast to context offers and chains, context requests only express their preferences regarding the metadata and cost dimensions, e.g. *Context Request 1* has a high preference to minimize cost and only a low preference to maximize the accuracy.



**Figure 9.1:** Motivating Example for the Selection

According to these preferences, it is a simple task to select a mediator chain for each context request separately and to ignore the preferences of the other context requests. This selection is highlighted by a darker color of the arrow. *Context Request 1* selects *Chain 2* as it is cheaper than *Chain 3* and *Chain 8*. In contrast *Context Request 2* chooses *Chain 5* as this chain provides the highest accuracy. *Context Request 3* requests a balanced ratio of cost and accuracy and thus uses *Chain 7* as the input provider.

But as shown in Figure 9.1, a context offer can serve as input provider not only for one chain, so e.g. *Context Offer 3* can be bound to *Context Request 1* via *Chain 8*, *Context Request 2* with *Chain 6*, and *Context Request 3* via *Chain 7*. Thus when evaluating a set of context offers respectively mediator chains regarding their usability, it is not useful to evaluate such a set independently for every request.

Sharing an offer would result in decreasing the total cost, which is a general requirement in our approach (see Requirement 7 in Section 1.2). For example, when using *Chain 6* and *Chain 7* as the input providers for *Context Request 2* and *Context Request 3* respectively, the total cost would be the actual cost for the context offer and the actual cost for both mediation chains:  $(50 - 50) + (50 - 50) + 50 = 50$ .

Also it is clear that when searching for an optimal solution for all queries, it is not possible to reuse simply the solutions found by the single selection method, as this can be contradictory to some other solutions. For example, a solution set containing *Chains*

2, 5, and 7 conflicts with the preference of *Context Request 1* to minimize the cost as much as possible. Consequently an alternative has to be found which minimizes the number of contradictions of the different requests as much as possible. In our example, this could be *Context Offer 3* with *Chains 6 – 8*. This is the optimal solution for *Context Request 3*, whereas *Context Request 1* and *Context Request 2* have to cut back with regard to accuracy maximization and cost minimization respectively. We will explain in detail how to calculate the most promising solution in the following sections.

## 9.2 The Selection Approach

In this section the selection approach will be introduced in a formal way. First the syntactic elements are introduced, followed by some constraints regarding the syntax and afterwards the semantics.

### 9.2.1 Syntactic Elements

Before formally explaining the selection process, the syntactic elements are introduced:

- The set  $CR$  is the set of context services which implement a reasoning mechanism and require additional context information. Other context services are in set  $CS$ . A context service can only be element of  $CR$  or  $CS$ , hence  $CR \cap CS = \emptyset$ .
- The set  $O_S$  is the set of context offers provided by context services. The cardinality of the set is  $|O_S| = n_S$ . Offers can either be provided by context reasoners or by regular context services. Let  $O_{CR} \subset O_S$  be the set of offers provided by context reasoners and  $O_{CS} \subset O_S$  be the set of offers provided by regular context service. Again offers can only be provided by either a context reasoner or by a regular context service, hence  $O_{CR} \cap O_{CS} = \emptyset$  and  $O_{CS} = O_S \setminus O_{CR}$ .
- The set  $R$  contains all context queries registered by the various context consumers. Its cardinality is  $|R| = n_R$ .  $R_{CR} \subset R$  is the subset of requests expressed by context reasoners. The complement of the set of all requests  $R$  and the set of requests expressed by reasoners  $R_{CR}$  builds the set of regular requests  $R_{CS} = R \setminus R_{CR} \implies R_{CR} \cap R_{CS} = \emptyset$ .
- $M_r^o$  is the set of all mediator chains mediating between an offer  $o \in O_S$  and the request  $r \in R$ . The set  $M$  contains all mediator chains for all offers in  $O_S$  and all request in  $R$ , thus  $M = \bigcup_{o \in O_S, r \in R} M_r^o$ . The set  $M^o$  is the set of all mediator chains using a context offer  $o \in O_S$  as input provider. Hence  $M^o = \bigcup_{r \in R} M_r^o$ . The set  $M_r$ , containing all the mediator chains that provide input to fulfil the context request  $r \in R$  is defined as  $M_r = \bigcup_{o \in O_S} M_r^o$ . The offer  $o_m^{in}$  is the offer of the input provider of the mediator chain  $m$ , whereas the offer  $o_m$  is the offer of the mediator chain itself, thus  $o_m^{in} \in O_S$ .
- $O_M$  is the set of context offers provided by mediator chains in  $M$ . As described in the matching chapter (especially in Section 8.2), this set does not necessarily contain an offer  $o \in O_S$ , hence an offer that is provided by a context service. The cardinality of  $O_M$  is  $|O_M| = n_M$ . The set  $O$  contains all original offers and offers of mediator chains, hence  $O = O_S \cup O_M$ .

- $D$  is the set of quality dimensions used in the different context offers and queries to express the metadata constraints and the selection function (e.g. accuracy, freshness, reliability). The cardinality of  $D$  is  $|D| = m$ .
- The function  $q(o, d) : O \times D \rightarrow [0, 1]$  maps a quality dimension  $d \in D$  in an offer  $o$ ,  $o \in O$  to the normalized quality value.
- $C$  is the set of cost dimensions used in the different context offers and queries to express the cost constraints and the preferences regarding the different cost dimensions in the selection function.
- The function  $c(o, d) : O \times C \rightarrow \mathbb{R}^+$  describes the cost for cost dimension  $d \in C$ , of a context offer  $o \in O$ .  $c_m(o, d)$  is the part of the cost for cost dimension  $d$  of an offer  $o \in O_M$  that is caused by the mediator chain  $m \in M$ , thus  $c_m(o, d) = c(o_m, d) - c(o_m^{in}, d)$ .
- Both functions  $q(o, d)$  and  $c(o, d)$  are calculated based on the domain of the respective quality alternatively cost dimension. The domain of an dimension  $d \in D \vee C$  is a function  $dom(o, d) : O \times (D \cup C) \rightarrow \mathbb{R} \times \mathbb{R}$ . This calculation is described in detail in Section 9.3.
- In order to calculate the domain of a quality or cost dimension for an offer  $o$ , the set of specified metadata constraints is used. In contrast to the previous chapters where constraints consist of a dimension, an operator, a representation, and a value, we can simplify the constraints here, as after the matching phase the representation information are not required any more. The set of operators is  $OP = \{EQ, NEQ, GT, NGT, LT, NLT, GE, NGE, LE, NLE\}$  (see Section 7.2). A constraint is a mapping  $con(o, op, d) : O \times OP \times (C \cup D) \rightarrow \mathbb{R}$ . All constraints used in an offer  $o \in O$  are contained in a set  $CON_o$ .
- As described in Section 7.3, the context consumer can optionally express his preferences regarding the different quality or cost dimensions. These values are used afterwards during the selection to find an optimal context provider. The variable  $w_{dr}$  is the preference/weight of a quality/cost dimension  $d \in D \vee C$  expressed in a context request  $r \in R$ . Additionally, the consumer can express preferences regarding cost in general and not only to specific cost dimensions. The variable  $w_{cr}$  describes the preference/weight regarding cost  $c$  in a context request  $r \in R$ .

Based on the previously introduced syntactic elements, the following syntactic constraints must hold:

1. The weight regarding a certain quality dimension  $d$  or the cost  $c$  has to be between 0 and 1:

$$0 \leq w_{dr} \leq 1 \quad (9.1)$$

$$0 \leq w_{cr} \leq 1 \quad (9.2)$$

2. The sum of all weights used in a selection function of a context request has to be 1:

$$\sum_{d \in D} w_{dr} + w_{cr} = 1 \quad (9.3)$$

## 9.2.2 Semantics

After introducing the syntactic elements, the actual selection based on the syntactic elements and the constraints will be explained. The first step of the selection process is the calculation of the quality and cost values.

**Quality Value Calculation** The function  $q(o, d) : O \times D \rightarrow [0, 1]$  calculates the quality value of an offer  $o \in O$  for a quality dimension  $d \in D$ . As the result of  $q(o, d)$  is normalized, we have to calculate the actual value before the normalization. For this purpose, we use the domain  $dom(o, d)$  of the dimension  $d$  in offer  $o$ , which is calculated based on the metadata constraints and on historical values. The result of this function is an interval  $(x, y)$ . The function is explained in detail in Section 9.3. At this place, we assume that  $dom(o, d) = (x, y)$ .

$$v(o, d) = \begin{cases} \frac{x + y}{2}, & \text{if } x \neq -\infty \wedge y \neq +\infty, \\ x, & \text{if } minPref = false \wedge x \neq -\infty \wedge y = +\infty, \\ y, & \text{if } minPref = true \wedge x = -\infty \wedge y \neq +\infty, \\ 0, & \text{else.} \end{cases} \quad (9.4)$$

$v(o, d)$  is calculated by Equation 9.4. In general we prefer the average value of a dimension. But this requires both lower and upper limit of the domain to be available, here  $x$  and  $y$ , and both limits to be finite. Otherwise, we try to retrieve the worst case. This depends on whether metadata of a certain type should be maximized, e.g. reliability in percent, or should be minimized like accuracy as a radius in meter.<sup>1</sup> Metadata that should be maximized, thus  $minPref = false$ , have in worst case the value of their lower limit,  $x$  in this case. Inversely, metadata that should be minimized, hence  $minPref = true$ , have in worst case the value of their upper limit,  $y$  in this case. If the required limit is infinite,  $v(o, d)$  is zero. This is necessary as the maximum of  $v(o, d)$  will be used for the normalization and should not be  $+\infty$ .

$$q(o, d) = \begin{cases} \frac{v(o, d)}{\left(\max_{o' \in O} v(o', d)\right) + \epsilon}, & \text{if } minPref = false \wedge x \neq -\infty, \\ 1 - \frac{v(o, d)}{\left(\max_{o' \in O} v(o', d)\right) + \epsilon}, & \text{if } minPref = true \wedge y \neq +\infty, \\ \epsilon, & \text{else.} \end{cases} \quad (9.5)$$

<sup>1</sup>Accuracy is an interesting example for the variation of the preference regarding minimization or maximization of this value depending on the representation of these metadata. Accuracy can be interpreted as discrepancy in form of a radius in meter. The smaller the radius, the better (more accurate) the position is. In contrast, accuracy as used in general linguistic usage should be maximized.

$q(o, d)$  as shown in Equation 9.5 normalizes the value of  $v(o, d)$  to be in the interval  $[0, 1]$ . With the calculation of  $q(o, d)$ , we remove the dependency from the maximization/minimization preference  $minPref$ . In the calculation of  $q(o, d)$  we distinguish between the two cases  $minPref = true$  and  $minPref = false$ . In case of  $minPref = false$ , the metadata should be maximized. For that reason it is best to get as close as possible to the maximum of all values for this dimension for the different offers. In case of the preferred minimization of the metadata ( $minPref = true$ ), it is to the preferred maximization contrary: the closer to the maximum of all values of the different offers the value is, the worse. The normalized value is subtracted from 1 to achieve this. As a consequence,  $q(o, d)$  is better the closer it is to 1. We check again during the calculations if the required limits are finite. Hence it is checked if the lower limit  $x$  is finite for metadata that should be maximized and if the upper limit  $y$  is finite for metadata that should be minimized. If this is not the case, so if  $(minPref = false \wedge x = -\infty)$  or  $(minPref = true \wedge y = +\infty)$ , then  $q(o, d) = \epsilon$  with  $\epsilon$  near 0. By using  $\epsilon$  instead of 0 at this place, we avoid in the later phase that a context offer is removed from the solution space even if it can only provide a really small contribution (which is nevertheless better than no contribution).

**Cost Value Calculation** In our approach we search for an optimal selection of context offers respectively mediator chains fulfilling a set of context requests. As after a successful matching of an offer and a request, one or more mediator chains are created. For these offer-request pairs that match without mediation,<sup>2</sup> we search within the set of offers provided by the mediation chains  $O_M$ . Consequently, the selected set of context offers is within the powerset of  $O_M$ .

During the selection, not only the context requests have to be satisfied. The selection process has also to struggle with the general requirement to minimize the resource consumption and other costs. To calculate an optimal set of offers that satisfies both the context requests and the system requirement, we have to take into account the aggregated cost for a set of context offers  $X \in \mathcal{P}(O_M)$ . The cost for an arbitrary set of context offers  $X$ , where  $X$  is an element of the powerset of the set of all offers provided by the chains  $O_M$ , is the sum of all cost dimensions of all offers  $o \in X$ .

The value for a cost dimension  $d \in C$  is calculated similarly to the value for other metadata dimensions with the difference that costs should never be maximized. Thus Equation 9.4 can be simplified to Equation 9.6 to calculate the value of the cost dimension. Here, we assume again that  $dom(o, d) = (x, y)$ .

$$c(o, d) = \begin{cases} \frac{x + y}{2}, & \text{if } x \neq -\infty \wedge y \neq +\infty, \\ y, & \text{if } x = -\infty \wedge y \neq +\infty, \\ 0, & \text{else.} \end{cases} \quad (9.6)$$

<sup>2</sup>For an offer and a request that match without any mediation, a mediator chain is created only containing an *Identity Mediator*. This chain only serves as an interconnecting/binding of the loosely coupled service and consumer.

For the calculation of  $c(o, d)$  the same conditions hold: the favoured solution is to calculate the average value. If this is not possible, as the lower and/or upper limit are infinite, the worst case is calculated (only if the lower limit is finite) or the result is 0.

To calculate the aggregated cost for a set of context offers, we first have to take care of the fact, that several mediator chains can have the same context service and hence the same context offer as input provider. We split the costs into costs that are caused by the context services respectively input providers and that are caused by the actual mediator chains. For that purpose we have to define the set of mediator chains  $M_X$  providing a set of context offers  $X \in \mathcal{P}(O_M)$ :

$$M_X = \{m | m \in M \wedge o_m \in X\} \quad (9.7)$$

Based on this definition, we can ascertain the set of context offers  $O_{M_X}$  serving as input provider for a set of mediator chains  $M_X$ :

$$O_{M_X} = \{o | o \in O_S \wedge \exists m \in M_X : o_m^{in} = o\} \quad (9.8)$$

With these two definitions, the value of a cost dimension  $d$  for a set of context offers provided by mediator chains  $c(X)$ ,  $X \in \mathcal{P}(O_M)$ , is calculated as follows:

$$c(X, d) = \sum_{o \in X} c_m(o, d) + \sum_{o \in O_{M_X}} c(o, d) \quad (9.9)$$

The first summand is the sum of the cost dimension  $d$  of the mediator chains providing the offers in  $X$ . The second summand is the sum of the cost dimension of the input providers of these chains.

As motivated before, we evaluate the usefulness of a set of context offers regarding the preferences mentioned in the context requests. Additionally, we introduced the requirement to minimize the total cost (see Requirement 7). This system requirement is also expressed by a utility function that has to be maximized. For this purpose, we introduce the function  $u_c(X, d)$ , which expresses the usefulness of a set of context offers  $X$  regarding resource respectively cost saving of cost dimension  $d$  in a scale between 0 and 1. A value of 0 is worst as this means a maximal cost production, whereas a value of 1 is best as it means that no costs are generated.

This utility for a set of context offers  $X \in \mathcal{P}(O_M)$  and a cost dimension  $d \in C$  is also called *Cost Minimization Utility Function (CMUF)* and is calculated as following:

$$u_c(X, d) = \begin{cases} \epsilon, & \text{if } \exists o \in X, \text{dom}(o, d) = (x, y) \wedge y = +\infty, \\ 1 - \frac{c(X, d)}{c(O_M, d)}, & \text{else.} \end{cases} \quad (9.10)$$

In this function,  $c(O_M, d)$  is the maximal value for the cost dimension  $d$  and hence the cost for all offers and mediation chains for that dimension. The utility  $u_c(X, d)$  for a set  $X$  of context offers and cost dimension  $d$  is maximal for an empty set ( $\implies u_c(X, d) = 1$ )



and minimal for the set of all offers and mediation chains ( $\implies u_c(X, d) = 0$ ). If it is not possible to determine the value for one of the offers in  $X$ ,  $u_c(X, d)$  becomes  $\epsilon$ . This might be the case if for one of the offers the upper limit is infinite and as a result neither the average nor the worst case value can be calculated. Missing information are therefore said to have the worst influence.

**Single Utility Function** The utility for cost dimensions  $u_c(X, d)$  and the valuation factor  $q(o, d)$  are now used to validate context offers regarding the different metadata and cost preferences mentioned in a context request. Based  $u_c(X, d)$  and  $q(o, d)$ , and with the preferences as expressed in a context request  $r \in R$  the utility of a set of context offers  $X \in \mathcal{P}(O_M)$  can be calculated as shown in Equation 9.12. This utility function is also called *Single Utility Function (SUF)*.

For the calculation of the SUF, we redefine the weights for the different cost dimensions. As described in Section 9.2.1, the context consumer can express a preference regarding cost in general and not only to specific cost dimensions with the variable  $w_{cr}$ . This weight is equally distributed about all weights for cost dimensions. For this purpose, the weights for cost dimensions are redefined as following:

$$w'_{dr} = w_{dr} + \frac{w_{cr}}{|C|} \text{ for } d \in C \quad (9.11)$$

With this redefined weight the SUF changes to:

$$u_r(X) = \begin{cases} 0, & \text{if } \neg(\exists m \in M^r : o_m \in X), \\ 0.5, & \text{if no selection function is defined for } r \wedge \exists m \in M^r : o_m \in X, \\ \left( \max_{x \in \{y \in X | \exists m \in M^r : o_m = y\}} \left( \sum_{d \in D} w_{dr} \cdot q(x, d) \right) \right) + \sum_{d \in C} w'_{dr} u_c(X, d), & \text{else.} \end{cases} \quad (9.12)$$

When calculating the utility of a set of context offers  $X$  regarding a request  $r$ , we distinguish between three cases:

- The utility is 0 if no mediator chain exists the offer of which is in  $X$  and that provides an appropriate output for the request. (This set is useless for the request  $r$ .)
- The utility is 0.5 if a mediator chain exist but no selection function has been specified. This specification is optional and means that the context consumer has no explicit preferences regarding QoC and CoC. As a consequence, the consumer accepts any context chain (that passes the matching process). This is also an important reason for the additional system requirement. The selection process ends in the cheapest option, if not other context request has influence on it.
- In the third case, the utility is calculated based on the preferences specified in the selection function. For this purpose, the weighted sum of the different quality dimensions is calculated as  $\sum_{d \in D} w_{dr} \cdot q(x, d)$  for every offer  $x \in$

$\{y \in X \mid \exists m \in M^r : o_m = y\}$ . This set only contains these offers from  $X$  that are context offers of mediator chains mediating between an input offer and the request  $r$ . As we are only searching for one context provider, we are selecting the one that provides the maximal result for the aforementioned condition. We then add the summand expressing the influence of the cost of the set. This summand is also a weighted sum  $\sum_{d \in C} w'_{dr} u_c(X, d)$ .

The utility  $u_r(X)$  expresses to what degree a set of offers  $X \subseteq O_M$  fulfils the preferences of a request  $r$ .

**Aggregated Utility Function** In our approach, a selection function is used to find a set of context offers fulfilling all context request as good as possible. This selection function is called *Aggregated Utility Function (AUF)*.<sup>3</sup> As shown in Equation 9.13, the AUF among others aggregates the SUF for every context request.

$$u(X) = \begin{cases} 0, & \text{if } \exists r \in R, u_r(X) = 0, \\ (1 - \alpha) \cdot \left( \sum_{r \in R} u_r(X) \right) + \alpha \cdot \frac{\sum_{d \in C} u_c(X, d)}{|C|}, & \text{else.} \end{cases} \quad (9.13)$$

In general, the AUF is a linear combination (more precisely a convex combination) of two factors: the first aggregates the SUFs and the second represents the average of the cost minimization utility function (CMUF) for all cost dimensions.

Given the SUF formulae, it is obvious that if a context request does not have weights on the cost dimensions, so  $w_{dr} = 0$  for every  $d \in C$  and  $w_{cr} = 0$ , every set that contains the offer with the maximal sum for the QoC values has the same utility. Ergo, it is important that the final selection – the AUF – has at least a minimal influence on these sets with regard to the cost minimization. This influence is controlled by the parameter  $\alpha$  in Equation 9.13.

The AUF  $u(X)$  is 0 if a request  $r$  exists that cannot be satisfied with this set  $X$ . Requests that cannot be satisfied by at least one offer, are already removed during the matching process as otherwise the AUF would be 0 for every set of offers. If every request can be satisfied with at least one offer in  $X$ , the function is calculated as described.

### 9.3 Calculation of the Domain of a Quality or Cost Dimension

This section described the calculation of the domain of a quality or cost dimension in an offer. The domain of a metadata dimension is the complete set of possible values of metadata of that dimension. As motivated in the previous section, this domain is used to estimate the average or at least worst case value of a quality or cost dimension in an offer. In general, the domain range of metadata or cost dimension  $d \in D \cup C$  in an context offer  $o \in O$  is a function  $dom : O \times (D \cup C) \rightarrow \mathbb{R} \times \mathbb{R}$ . Thus the domain is expressed as an interval.

<sup>3</sup>The Aggregated Utility Function is also called Aggregated Objective Function in literature, e.g. in [94].

The metadata constraints  $CON_o$  used in an offer  $o$  serve as a starting point for the calculation of the domain. We define two subsets of the set of operators  $OP$ : a set of operators to retrieve the lower limit of the domain  $OP_L = \{EQ, NLT, GE, GT, NLE\} \subset OP$  and a set to retrieve the upper limit of the domain  $OP_U = \{EQ, NGT, LT, NGE, LE\} \subset OP$ .

With these subsets a first estimation  $dom'$  of the domain for an offer  $o$  and a cost or quality dimension  $d$  can be defined as expressed in Equation 9.14.

$$dom'(o, d) = \begin{cases} (\min \{y | \exists op \in OP_L : con(o, op, d) = y\} , \\ \max \{y | \exists op \in OP_U : con(o, op, d) = y\}) \\ , \text{ if } \exists op_1 \in OP_L : con(o, op_1, d) \in CON_o \\ \wedge \exists op_2 \in OP_U : con(o, op_2, d) \in CON_o, \\ \\ (-\infty, \max \{y | \exists op \in OP_U : con(o, op, d) = y\}) , \\ \text{if } \nexists op_1 \in OP_L : con(o, op_1, d) \in CON_o \\ \wedge \exists op_2 \in OP_U : con(o, op_2, d) \in CON_o, \\ \\ (\min \{y | \exists op \in OP_L : con(o, op, d) = y\} , +\infty) , \\ \text{if } \exists op_1 \in OP_L : con(o, op_1, d) \in CON_o \\ \wedge \nexists op_2 \in OP_U : con(o, op_2, d) \in CON_o, \\ \\ (-\infty, +\infty) , \text{ else.} \end{cases} \quad (9.14)$$

With this equation the domain of a metadata dimension  $d$  in an offer  $o$  is defined as an interval. If one or more constraints exist limiting the set upwards, the operator of this constraint has to be in set  $OP_U$ . The upper limit of this interval is the maximum of the values used in these constraints. Otherwise the upper limit is infinite. The lower limit of the domain interval is defined respectively: if one or more constraints exist limiting the set downwards, the operator of this constraint has to be in set  $OP_L$ . The lower limit of this interval is the minimum of the values used in these constraints. Otherwise the lower limit is negative infinite.

The equation presented in Equation 9.14 is the calculation of the domain of a cost or metadata dimension without any historical data only based on the metadata constraints. These constraints only roughly limit the domain. As one main requirement on our system is the support for activation and deactivation of required and not required local context provider (see Requirement 5), it is in general not possible to request the context provider for its current status and hence for up-to-date metadata. Using a collection of historical data to estimate a domain is much more insusceptible with regard to outlier as using only a single value even if it is much more up-to-date. For these reasons, historical values can provide an important feedback on the current status of the context service and can be helpful to make the domain definition more precise. Nevertheless, historical values do not necessarily reflect the current status of its provider.

We define the set of historical data as a set of tuples consisting of the metadata value  $v$  and the timestamp  $t$ : the set of historical data for dimension  $d$  of offer  $o$  is  $H'_{od} = \{(v_1, t_1), (v_2, t_2), \dots, (v_n, t_n)\}$  with  $v_i$  is value of dimension  $d$  at time  $t_i$ . Furthermore we

define current time as  $t_0$ . To this set of historical data we add the previously calculated upper and lower limit of the domain. As we assume constraints to be permanently valid, the current time is assigned to both limits as timestamps:

Assume  $dom'(o, d) = (x, y)$ , then  $H = H' \cup \{(v_{n+1} = x, t_0), (v_{n+2} = y, t_0)\}$ .

Context data and also their metadata change permanently. As a result, metadata are getting more and more uninteresting for the calculation of the domain the older they are. In order to calculate the new domain of the metadata type based on historical data, we discount the different values based on their age  $t_0 - t_i$ . For this calculation we introduce a new system parameter  $\beta$ , which represents the half-value period of metadata in seconds. This means after  $\beta$  seconds the influence respectively importance of the metadata is halved: after 0 seconds the importance is 1, after  $\beta$  seconds the importance is  $\frac{1}{2}$ , after  $2\beta$  seconds the importance is  $\frac{1}{4}$ , and so on:

$$\bar{w}_i = \frac{1}{2^{x_i}} \text{ with } x_i = \frac{t_0 - t_i}{\beta} \quad (9.15)$$

In this analysis we assume newer values to have a higher relative frequency than older values. As a result, the weight  $\bar{w}_i$  is interpreted as influencing factor or relative frequency. Values with a high weight should have a high influence respectively frequency on the domain interval, whereas values with a low weight should have low influence respectively frequency. Consequently our domain interval should preferably include values with high influence respectively frequencies. In descriptive statistics, the cumulative frequency analysis is used for this. A cumulative frequency is the aggregation of all relative frequencies of these values which fall at or below a given value. For this analysis we have to determine a relative frequency distribution from the weights by normalizing it by dividing by the sum of all values:

$$w_i = \frac{\bar{w}_i}{\sum_{z=1}^{n+2} \bar{w}_z} = \frac{\bar{w}_i}{\sum_{z=1}^n \bar{w}_z + 2} \quad (9.16)$$

With this equation, we can define a new set of tuples containing the metadata value with the calculated weight. To calculate the domain, this set is ordered in ascending order of values:

$$\begin{aligned} & (v'_1, w'_1) \dots (v'_i, w'_i) \dots (v'_{n+2}, w'_{n+2}) \\ \text{with } & v'_i \in \{v_1, \dots, v_n, x, y\} \wedge v'_i \leq v'_{i+1} \\ & \wedge (v'_i = v_j \wedge w'_i = w_j, 1 \leq j \leq n+2) \end{aligned} \quad (9.17)$$

With this ordered set, we can now define the domain of a metadata dimension  $d$  in an offer  $o$  as following:

$$dom(o, d) = \begin{cases} dom'(o, d), & \text{if } H_{od} = \emptyset, \\ \left( v'_{\min\left\{x \mid \left(\sum_{i=1}^x w'_i\right) \geq 0.25\right\}}, v'_{\min\left\{x \mid \left(\sum_{i=1}^x w'_i\right) \geq 0.75\right\}} \right), & \text{else.} \end{cases} \quad (9.18)$$

From the equation it becomes clear that if no historical data are available our domain is the interval calculated by Equation 9.14. Otherwise the lower limit is defined by the lower quartile of the cumulative frequency and the upper limit by the upper quartile of the cumulative frequency.

The usage of the lower and upper quartile is motivated by the rather common approach in statistics to define a range as its interquartile range (IQR) instead of the difference between the highest and lowest values. As stated by Upton et al. [136], the usage of the interquartile range has the advantage that it does not ignore the distribution of the intermediate values. Upton et al. further mention that “a single very large or very small value would give a misleading impression of the spread of the data”.

The resulting domain can be informally described as approximately 75% of all values are above the lower limit and approximately 75% are below the upper limit.

### 9.3.1 Example

This subsection explains the calculation of the domain of the metadata type ‘accuracy’ for an offer  $o$ . We initially have to estimate the domain based on the metadata constraints. For this purpose, we have to calculate  $dom'(o, accuracy)$  as described in Equation 9.14. The metadata constraints of offer  $o$  are shown in Listing 9.1.

```

1 IContextOffer offer = coqlFactoryImpl.eINSTANCE.
  createContextOffer ();
2 [...]
3 IAtomicNumericalMetadataConstraint con1 = coqlFactoryImpl.
  eINSTANCE.createAtomicNumericalMetadataConstraint ();
4 con1.setMetadata("#Accuracy");
5 con1.setRepresentation("#DoubleRep");
6 con1.setUnit("#meter");
7 con1.setMetadataConstraintID("Accuracy1");
8 con1.setOperator(RelationalOperator.LT_LITERAL);
9 con1.setValue(Factory.createValue(200));
10 offer.getMetadataConstraints().add(con1);
11 IAtomicNumericalMetadataConstraint con2 = coqlFactoryImpl.
  eINSTANCE.createAtomicNumericalMetadataConstraint ();
12 con2.setMetadata("#Accuracy");
13 con2.setRepresentation("#DoubleRep");
14 con2.setUnit("#meter");
15 con2.setMetadataConstraintID("Accuracy2");
16 con2.setOperator(RelationalOperator.GT_LITERAL);
17 con2.setValue(Factory.createValue(5));
18 offer.getMetadataConstraints().add(con2);

```

**Listing 9.1:** Example of Metadata Constraints 3

With these constraints, the result of the calculation is  $dom'(o, accuracy) = (5, 200)$ . In order to precise this estimation, we reuse historical values:<sup>4</sup>

$$H'_{o,accuracy} = (42, 1113757440) \dots (400, 1113757145)$$

<sup>4</sup>In our implementation we use the Unix timestamp as the internal representation for timestamps. The Unix timestamp is defined as seconds elapsed since midnight Coordinated Universal Time (UTC) of January 1, 1970. [http://en.wikipedia.org/wiki/Unix\\_time](http://en.wikipedia.org/wiki/Unix_time). Last visited on Mar 13, 2012

As mentioned earlier, we add both values of the estimated domain  $dom'$  with the current time  $t_0 = 1113757445$  to this set. The resulting set of tuples  $H$  is shown in Table 9.1.

<b>Sample i</b>	1	2	3	4
<b>Accuracy acc (m)</b>	42	7	12	55
<b>Time t</b>	1113757440	1113757325	1113757345	1113757355
<b>Sample i</b>	5	6	7	8
<b>Accuracy acc (m)</b>	3	21	8	400
<b>Time t</b>	1113757245	1113757415	1113757385	1113757145
<b>Sample i</b>	9	10		
<b>Accuracy acc (m)</b>	5	200		
<b>Time t</b>	1113757445	1113757445		

**Table 9.1:** Example for Domain Calculation – Historical Values

With this set we can now calculate the new domain as described above. Before the calculation we have to initially define a half-value period. In this example we use a half-value period of  $\beta = 60$  seconds.

In Table 9.2 the results of the different intermediate steps for the calculation of the domain are shown.<sup>5</sup> First, the age of the metadata instance has to be calculated. With the age the discounting factor  $\bar{w}$  can be calculated, which is afterwards normalized. The normalization result  $w$  is then used to define the cumulative frequency.

With these results, the new upper and lower limits can be determined: the lower limit is the first value, the cumulative frequency of which is higher than or equal to 0.25; the upper limit is the first value, the cumulative frequency of which is higher than or equal to 0.75. As a result, the domain is  $dom'(o, accuracy) = (7, 55)$ , which is much more precise than the domain defined only by the constraints  $dom'(o, accuracy) = (5, 200)$ .

## 9.4 Selection Algorithm

In the previous sections the functions to evaluate the usefulness of a set of context offers  $X \subset O_M$  regarding a set of context requests have been described. For this purpose, the Aggregated Utility Function (AUF) is calculated. Evaluating every subset of the set of offer  $O_M$  causes exponential computational costs. The powerset  $\mathcal{P}(O_M)$  of  $O_M$  (the set of all subsets of  $O_M$ ) contains  $2^{O_M}$  sets. However, most of these subsets would only cause the calculation of the AUF to deliver a result of 0 as one of the requests cannot be fulfilled. For example, a set containing only an offer fulfilling one request but not all requests, would result 0 for the AUF. Hence, avoiding the calculation of the AUF for these useless subsets substantially reduces the computational costs and improves scalability. A subset of  $O_M$  which can potentially result in an AUF  $> 0$  has only to fulfil the following requirement: It has to contain exactly one offer for every query which

<sup>5</sup>Please note here that in this table the tuples are already sorted to ascending accuracy values.

i	5	9	2	7	3
i'	1	2	3	4	5
acc	3	5	7	8	12
t	1113757245	1113757445	1113757325	1113757385	1113757345
age	200	0	120	60	100
$\bar{w}$	0.099	1.000	0.250	0.500	0.315
w	0.019	0.192	0.048	0.096	0.061
$\sum_{z=1}^{i'} w$	0.019	0.211	0.259	0.356	0.416
i	6	1	4	10	8
i'	6	7	8	9	10
acc	21	42	55	200	400
t	1113757415	1113757440	1113757355	1113757445	1113757145
age	30	5	90	0	300
$\bar{w}$	0.707	0.944	0.354	1.000	0.031
w	0.136	0.182	0.068	0.192	0.006
$\sum_{z=1}^{i'} w$	0.552	0.734	0.802	0.994	1.000

**Table 9.2:** Example for Domain Calculation – Intermediate Results

can serve as a input provider for that query. Subsets containing no offer for one of the queries would lead to an AUF of 0. A subset containing more than one offer per query would not be selected as well. More than one offer per query has no benefit (as we do not provide any context fusion support) but increases the cost. Such a set would result in a lower AUF as every subset containing only one of the offers for that query due to the aggregation of the cost. Consequently we use the algorithm as described in Algorithm 9.1 to calculate the subsets of  $O_M$ , which potentially build the optimal selection for the different queries. In this calculation we have to distinguish between queries send from context reasoners and queries send from other context consumers. Context queries from context reasoners only have to be satisfied by the selection set if also a chain using the context reasoner as input provider is part of this set. In contrast context requests from other consumers have to be fulfilled by every potential selection set. As shown in line 1 and 2 of the algorithm, context queries are distributed over the two arrays  $query[]$  and  $reasonerQuery[]$ . A context reasoner in general provides a context offer and can request one or more different types of context information. In order to retrieve the corresponding context queries of a reasoner that is used in a chain, a map is established mapping an offer of a reasoner to a set of queries (of that reasoner, see line 4).

---

**Algorithm 9.1** Algorithm  $select(query[], chainSet)$  to select a set of mediator chains from all chains  $chainSet$  for the array of context queries  $query[]$

---

- 1:  $query[]$  ▷ array of all context queries without queries from reasoners
- 2:  $reasonerQuery[]$  ▷ array of all context queries from reasoners
- 3:  $chainArrayForQueryMap$  ▷ global variable that maps a query to an array containing all chains useful as input provider for that query; used  $chainSet$  for

```

initialization
4: reasonerQueryForOfferMap  $\triangleright$  global variable that maps an offer to a set of
   queries required by a reasoner
5: mapOfQueryIndex  $\triangleright$  global variable mapping a query to an integer (initially to 0)
6: numOfComb  $\leftarrow$  1
7: for  $x \leftarrow 0; x < query.length; x+ = 1$  do
8:   numOfComb  $\leftarrow$  numOfComb  $\cdot$  chainsForQueryMap[query[ $x$ ]].size
9: end for
10: utilityOfSelectionSet  $\leftarrow$  0.0
11: Set  $\langle$  IMediatorChain  $\rangle$  selectionSet  $\leftarrow$  Set  $\langle$  IMediatorChain  $\rangle$  ()
12: for  $z \leftarrow 1; z \leq numOfComb; z+ = 1$  do
13:   Set  $\langle$  IMediatorChain  $\rangle$  combination  $\leftarrow$  Set  $\langle$  IMediatorChain  $\rangle$  ()
14:   Set  $\langle$  IContextQuery  $\rangle$  reqReasonerQueries  $\leftarrow$  Set  $\langle$  IContextQuery  $\rangle$  ()
15:   increaseFlag  $\leftarrow$  true
16:   for  $y \leftarrow 0, query.length$  do
17:      $i \leftarrow mapOfQueryIndex[query[y]]$ 
18:      $chain \leftarrow chainsForQueryMap[query[y]][i]$ 
19:     if reasonerQueryForOfferMap.contains(chain.inputOffer) then
20:       reqReasonerQueries.addAll(reasonerQueryForOfferMap.get(
        chain.inputOffer))
21:     end if
22:     combination.add(chain)
23:     if increaseFlag then
24:        $i \leftarrow i + 1$ 
25:       if  $i \geq chainsForQueryMap[query[y]].size$  then
26:          $mapOfQueryIndex[query[y]] \leftarrow 0$ 
27:          $i \leftarrow 0$ 
28:         for  $s \leftarrow y + 1; s < query.length; s+ = 1$  do
29:            $i2 \leftarrow mapOfQueryIndex[query[s]] + 1$ 
30:           if  $i2 < chainsForQueryMap[query[s]].size$  then
31:              $mapOfQueryIndex[query[s]] \leftarrow i2$ 
32:             break
33:           else
34:              $mapOfQueryIndex[query[s]] \leftarrow 0$ 
35:           end if
36:         end for
37:       else
38:          $mapOfQueryIndex[query[y]] \leftarrow i$ 
39:       end if
40:       increaseFlag  $\leftarrow$  false
41:     end if
42:   end for
43:   reasonerComb  $\leftarrow$  1
44:   for  $x \leftarrow 0; x < reqReasonerQueries.length; x+ = 1$  do
45:     reasonerComb  $\leftarrow$  reasonerComb  $\cdot$  chainsForQueryMap[query[ $x$ ]].size
46:   end for
47:   for  $y \leftarrow 1; y \leq reasonerComb; y+ = 1$  do
48:     Set  $\langle$  IMediatorChain  $\rangle$  reasonerCombination  $\leftarrow$  Set  $\langle$ 

```



```

    IMediatorChain > ()
49:     for chain : combination do
50:         reasonerCombination.add(chain)
51:         if reasonerQueryForOfferMap.contains(chain.inputOffer) then
52:             increaseFlag2 ← true
53:             rQuery[] ← reasonerQueryForOfferMap.get(chain.inputOffer)
54:             for q ← 0; q ≤ rQuery; q+ = 1 do
55:                 i2 ← mapOfQueryIndex[rQuery[q]]
56:                 chain2 ← chainsForQueryMap[rQuery[q]][i2]
57:                 reasonerCombination.add(chain2)
58:                 if increaseFlag then
59:                     ...           ▷ Similar to increasing of pointers as above.
60:                 end if
61:             end for
62:         end if
63:     end for
64:     combinationUtility ← aggregatedUtilityFunction(reasonerCombination)
65:     if combinationUtility > utilityOfSelectionSet then
66:         utilityOfSelectionSet ← combinationUtility
67:         selectionSet ← reasonerCombination
68:     end if
69: end for
70: end for
71: return selectionSet

```

---

The set of all chains can be divided into subsets according to the usefulness for one request. The set of offers provided by context services can be useful for more than one request. In contrast, a context offer provided by a mediator chain  $o \in O_M$  is only useful for the request for which the chain has been constructed (see Section 8.2). Before starting the actual selection, we build a mapping from a query to an array of these context offers respectively mediator chains that are constructed for it (line 3). With this map we can easily calculate a valid subset of context offers respectively chains resulting in an AUF  $> 0$ . Additionally, we initialize a new map serving as indexes for every query (line 5).

These indexes are increased stepwise and serve as a pointer in the array of context chains of the respective context query.<sup>6</sup> For example, at the beginning (when calling the function) all indexes are set to 0. Consequently, for every query the first mediator chain is chosen and combined to a new set (line 22). Afterwards one of these indexes is increased (line 23–41). In the next run of the main loop (line 12–70), the second chain is chosen for the first query, whereas for all the other queries (still) the first is taken. The main loop consists of two internal loops calculating a valid chain set. The first loop (line 16–42) selects a chain for every query, which is not expressed by a reasoner. For every selected chain it is checked whether a context reasoner serves as input provider for that chain or not (see line 19–20). In the second loop (line 47–69) the set of chains

<sup>6</sup>This method is comparable to the systematic approach to pick a combination lock with the only difference that every ring (in our case query) can have a different cardinality of numbers (here mediator chains).

calculated by the first loop is extended by chains for every query of a used reasoner. These chains are chosen similarly to the first loop by increasing an index (line 58–60). For every of these combinations calculated in the second loop, the AUF is calculated (line 64) and the combination with the highest result is the selection set (line 66–67).

The outer loop (starting in line 12) is called in total  $numOfComb$  times, which is approximately the average number of mediator chains per regular (not from a reasoner expressed) query raised with the number of these queries. The first inner loop (starting in line 16) is used to retrieve the next combination of mediator chains for the regular queries and hence it is called  $|O_{CS}|$  times per run of the outer loop. The second inner loop (starting in line 47) calculates the different allocations of reasoner queries with appropriate chains. These different allocations can also be seen as different variants of a context reasoner. A variant of a context reasoner is a concrete allocation of its context queries with a chain. Hence a context reasoner with one query and with two potential chains for this query results in two variants (one using the first chain for the query and one using the second chain for the query).

In general, the number of variants of a reasoner can be approximated by the average number of mediator chains per query of that reasoner raised with the number of queries expressed by this reasoner. With this definition an offer provided by a context reasoner can be considered as a kind of abstract offer which is substantiated to a concrete offer by the concrete allocations. Hence a reasoner provides as many concrete offers as variants.  $O_{CR}$  is the set of context offers provided by context reasoner. Let  $O_r \subset O_{CR}$  be the offer provided by a context reasoner  $r \in CR$ . Similarly,  $R_{CR} \subset R$  is the subset of requests expressed by context reasoners. Let  $R_r \subset R_{CR}$  be the subset of requests expressed by a context reasoner  $r \in CR$ . In worst-case, there are mediator chains between every offer  $o \in O_S \setminus O_r$  and every query  $r \in R_r$ . In this case, we have

$$\left( \frac{|O_S \setminus O_r| \cdot |R_r|}{|R_r|} \right)^{|R_r|} = |O_S \setminus O_r|^{|R_r|} \text{ variants of that reasoner } r \in CR.$$

In the first part of the main loop we handle requests that are not expressed by context reasoners. The complement of the set of all requests  $R$  and the set of requests expressed by reasoners  $R_{CR}$  then builds the set  $R_{CS}$  of what we called regular requests before. The worst case can be approximated here similarly by assuming that we have a mediator chain from every offer  $o \in O_S$  to every query  $r \in R_{CS}$ , hence  $|O_S| \cdot |R_{CS}|$  chains. As defined in Section 9.2.1 an offer is either an element of the offers provided by context reasoners or by regular services, hence  $|O_S| = |O_{CR}| + |O_{CS}|$ . However, an offer of a context reasoner can be seen as some kind of abstract context offer which can be replaced by one or more concrete instances, which are concretely allocated chains to the different queries of that reasoner. A reasoner  $r \in CR$  can have  $(|O_S \setminus O_r|)^{|R_r|}$  variants, thus  $O_{CR}$  can be estimated by  $\sum_{r \in CR} (|O_S \setminus O_r|)^{|R_r|} \leq (|CR| \cdot |O_S|)^{|R_{CR}|}$ . The worst case complexity of the whole algorithm can so be approximated by  $(|O_{CS}| + (|CR| \cdot |O_S|)^{|R_{CR}|})^{|R_{CS}|}$ . Hence the algorithm has exponential complexity.

## 9.5 Example

This section explains the selection process using an elaborate example. Figure 9.2 shows three context queries and four context offers. In this example we assume that all offers

and queries passed the matching process and that we can ignore their representations here. However not all offers fit to every query:

- Offer 1 matches all queries, as it provides both the position and the altitude
- Offer 2 and 3 match only queries 1 and 2 but not query 3, as both do not provide the altitude.
- Offer 4 only satisfies query 3, as it provides only information on the altitude but not on the position.

Similar to the matching example in Section 8.4, the offers contain several clouds expressing the metadata constraints but we removed the representation here to improve the readability. The context queries in Figure 9.2 do not contain any metadata constraints, as these are not used in the selection phase but rather in the matching phase. Instead, they contain several selection function factors (boxes with the white hand in the left top corner). These factors express the preferences of the context consumer regarding QoC and CoC. For example, query 1 in Figure 8.4 contains two selection function factors, the first regarding accuracy and the second regarding cost. With these factors, the consumer defines that during the selection phase the only decision criteria is accuracy (indicated by the weight of 1.00 for accuracy) while cost is a minor factor (indicated by the weight of 0.00). Furthermore, all selection function factors contain an arrow in the top right corner, which shows the *minPref* and hence whether it is better to search for the minimum (arrow to the bottom) or to search for the maximum (arrow to the top) within the set of metadata.

In this example, we focus on the actual selection. All context offers contain only one metadata constraint per metadata dimension (e.g. only one constraint referring to accuracy). In addition, all constraints are limiting the respective metadata upwards and all metadata should be minimized (as indicated by the arrows in the queries). These upper limits represent the worst case and are essential for the calculation of quality respectively cost values (as explained in Section 9.2.2 and Section 9.2.2). Furthermore, it improves the readability of the example, as the values depicted in the figure are also used in the following calculations (more precisely the depicted values are the result of  $v(o, d)$  respectively  $c(o, d)$ ).

$v(o, d)$  respectively  $c(o, d)$  are used within the calculation of  $q(o, d)$  and  $u_c(X, d)$ . These functions map an offer and a cost respectively quality dimension to a value between 0 and 1 and provide the input for the single utility functions (SUF). As all quality and cost dimensions should be minimized,  $q(o, d)$  and  $u_c(X, d)$  are calculated as follows:

$$q(o, d) = 1 - \frac{v(o, d)}{\left(\max_{o' \in O} v(o', d)\right) + \epsilon} \quad \text{and} \quad u_c(X, d) = 1 - \frac{c(X, d)}{c(O_M, d)}.$$

For the calculation we have to identify the maximum for every quality dimension:

- $max_{accuracy} = 10000$
- $max_{accuracy\_alt} = 50$

Consequently,  $q(o, d)$  for **Offer 1** and the quality dimension *accuracy* is  $q(o1, acc) = 1 - \frac{10}{10000 + \epsilon} = 0.9990$ .

Similarly, we have to identify the maximum for every cost dimension. In contrast to the quality dimensions this is not the highest value of  $v(o, d)$  respectively  $c(o, d)$  but rather the aggregated cost caused by all context offers together:

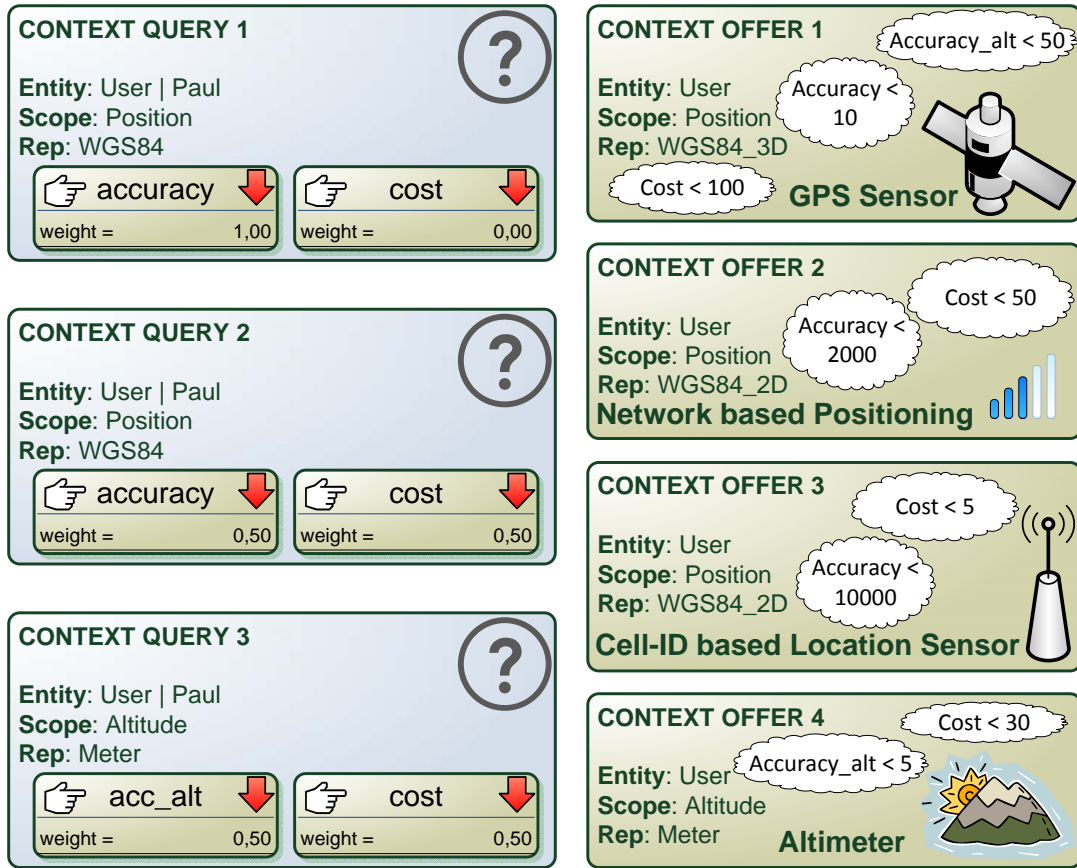


Figure 9.2: Example of a Selection Process – Offers and Queries

- $max_{cost} = c(O_M, d) = 100 + 50 + 5 + 30 = 185$

Hence,  $u_c(X, d)$  for a set  $X$  only containing Offer 1 and the cost dimension  $cost$  is  $u_c(o1, cost) = 1 - \frac{100}{185} = 0.4596$ .

The values can be calculated similarly for all dimensions and all different sets of offers and are afterwards used for the calculation of the SUFs. The calculation of the SUFs is the first part of the selection. A SUF provides a feedback on the usability of a set of context offers with regard to a single context request. The SUF  $u_r(X)$  for a set of offers  $X \subseteq O_M$  is a sum of two summands: the first summand  $\max_{x \in \{y \in X | \exists m \in M^r : o_m = y\}} \left( \sum_{d \in D} w_{dr} \cdot q(x, d) \right)$  expressing the utility of the ‘best’ offer within this set  $X$  with regard to quality dimensions. The second summand  $\sum_{d \in C} w'_{dr} u_c(X, d)$  represents the utility with regard to costs of the complete set. We only search for one context offer per request, as we do not support context fusion. Hence only one, actually the offer with the highest utility with regard to quality is considered.

The result of  $\max_{x \in \{y \in X | \exists m \in M^r : o_m = y\}} \left( \sum_{d \in D} w_{dr} \cdot q(x, d) \right)$  is presented for every combination of offers and queries in Table 9.3.

		Query 1	Query 2	Query 3
	Accuracy	1	0.5	0
	Cost	0	0.5	0.5
	Accuracy_alt	0	0	0.5
<b>Offer 1</b>		✓	✓	✓
Accuracy	10	0.9990	0.4995	-
Cost	100			
Accuracy_alt	50	-	-	0.0098
<b>Offer 2</b>		✓	✓	/
Accuracy	2000	0.8000	0.4000	/
Cost	5			/
Accuracy_alt	-	-	-	/
<b>Offer 3</b>		✓	✓	/
Accuracy	10000	0.00009	0.00005	/
Cost	5			/
Accuracy_alt	-	-	-	/
<b>Offer 4</b>		/	/	✓
Accuracy	-	/	/	-
Cost	30	/	/	
Accuracy_alt	5	/	/	0.4510

**Table 9.3:** Selection Example 1 – Intermediate Results

Based on these results, the complete SUFs and the aggregated utility function (AUF) can be calculated. The results are shown in Table 9.4. In this example we use  $\alpha = 0.05$  to express that mainly the context queries take care of cost minimization.

Table 9.4 can be read as follows: the four columns o1 - o4 are used to describe all subsets of the set of context offers  $X$  – the powerset of  $X$ . The fifth column ( $\sum cost$ ) contains the aggregated cost for the respective set of context offers, e.g. the set consisting of all four offers causes cost of  $100 + 50 + 5 + 30 = 185$ . In the next column  $u_c$  is calculated as described above in Equation 9.10. In the next three columns the SUFs of the single requests regarding the respective set of context offers are calculated. Therefore, the results of Table 9.3 are used. As in query 1 the preference regarding cost is 0, the utility of a set of offers is the maximum of the results of the respective subset of offers in Table 9.3. The last column contains the AUF, which is computed as described in Equation 9.13. For example in the third line for the set only consisting of offer 4 the AUF is 0, as this set cannot satisfy query 1 and query 2. Here it has to be highlighted that we calculated the SUFs/AUFs for the complete powerset of  $X$  only for this example. The rows resulting in a AUF of 0 are not calculated as they would not be a valid combination combined by the Algorithm 9.1. The maximum result (2.15) has been calculated for the set consisting of offers 1 and 4 (see line 10). This set satisfies all three requests without one of them has

#	o1	o2	o3	o4	$\sum cost$	$u_c$	$u_{q1}$	$u_{q2}$	$u_{q3}$	AUF
1					0	1	-	-	-	0
2				✓	30	0.84	-	-	0.87	0
3			✓		5	0.97	0.00	0.49	-	0
4			✓	✓	35	0.81	0.00	0.41	0.86	1.25
5		✓			50	0.73	0.80	0.76	-	0
6		✓		✓	80	0.57	0.80	0.68	0.73	2.14
7		✓	✓		55	0.70	0.80	0.75	-	0
8		✓	✓	✓	85	0.54	0.80	0.67	0.72	2.11
9	✓				100	0.460	1.00	0.73	0.23	1.88
10	✓			✓	130	0.30	1.00	0.65	0.60	2.15
11	✓		✓		105	0.43	1.00	0.72	0.22	1.86
12	✓		✓	✓	135	0.27	1.00	0.63	0.59	2.12
13	✓	✓			150	0.19	1.00	0.59	0.09	1.61
14	✓	✓		✓	180	0.03	1.00	0.51	0.46	1.88
15	✓	✓	✓		155	0.16	1.00	0.58	0.08	1.59
16	✓	✓	✓	✓	185	0	1.00	0.50	0.45	1.85

**Table 9.4:** Selection Example 1 – Results

to cut back. If we change, e.g. the weights in context query 1 to  $weight_{accuracy} = 0$  and  $weight_{cost} = 1$ , the result would look completely different. This is shown in Table 9.5. We can skip the calculation of the intermediate results, as the only change in Table 9.3 would be to replace the values for query 1 and the respective accuracy of the offers with – due to the full weight of this request on cost and not on QoC.

In this example it becomes clear that it is not always possible to fulfil all preferences. The set with the maximal AUF ( $AUF = 2.01$ , see line 4 in Table 9.5) is the set consisting of offers 3 and 4 (see line 4). This set does actually not select the optimal solution for all single requests. For query 1 the optimal selection would be offer 3 as it is the cheapest offer providing the requested information. For query 2, the set consisting of offer 2 would be the optimal solution and for query 3 the set consisting of offer 4. However, all requests have to cut back: offer 1 and 3 with regard to cost, offer 2 with regard to accuracy.

Another interesting observation is the effect caused by changing  $\alpha$ . In the previous calculation we have used  $\alpha = 0.05$ . A change to  $\alpha = 0.5$  would not result in another selection in the previous example (Table 9.5) as also the requests have an high interest on minimizing the costs, but it would result in another selection for the first example (Table 9.5), as depicted in Table 9.6.

Compared to Table 9.4 only the last column containing the AUF is changed. As shown in Table 9.6 the set of context offers causing the highest AUF ( $AUF = 1.39$ , see line 6 in Table 9.6) is the set consisting of offers 2 and 4. In comparison to the previous selection this is a drawback for request 1 as it requests high accuracy. However, this selection

#	o1	o2	o3	o4	$\sum cost$	$u_c$	$u_{q1}$	$u_{q2}$	$u_{q3}$	AUF
1					0	1	-	-	-	0
2				✓	30	0.84	-	-	0.87	0
3			✓		5	0.97	0.97	0.49	-	0
4			✓	✓	35	0.81	0.81	0.41	0.86	2.01
5		✓			50	0.73	0.73	0.76	-	0
6		✓		✓	80	0.57	0.57	0.68	0.73	1.91
7		✓	✓		55	0.70	0.70	0.75	-	0
8		✓	✓	✓	85	0.54	0.54	0.67	0.72	1.86
9	✓				100	0.460	0.46	0.73	0.23	1.37
10	✓			✓	130	0.30	0.30	0.65	0.60	1.48
11	✓		✓		105	0.43	0.43	0.72	0.22	1.32
12	✓		✓	✓	135	0.27	0.27	0.63	0.59	1.43
13	✓	✓			150	0.19	0.19	0.59	0.09	0.84
14	✓	✓		✓	180	0.03	0.03	0.51	0.46	0.95
15	✓	✓	✓		155	0.16	0.16	0.58	0.08	0.79
16	✓	✓	✓	✓	185	0	0.00	0.50	0.45	0.90

**Table 9.5:** Selection Example 2 – Results

causes fewer costs. Hereby it becomes obvious that the value of  $\alpha$  can be used to adapt the complete system e.g. to save resources when running out of battery.

## 9.6 Discussion

In this chapter, we introduced the selection process for searching an optimal set of context offers respectively context chains to be activated to satisfy all requests and also the system requirement regarding cost minimization. Several solutions especially in the area of service-oriented computing exist (like the approaches by Jaeger [62] and Yang et al. [148]) that separately select a service provider per request. In contrast to these context source and service selection approaches, we are searching for a solution which fulfils all requests and does not search for a solution for every query independently. As a result we use an aggregated utility function.

This has the advantage that it is possible not only to satisfy the requests but also to reduce the resource consumption, as this approach also facilitates sharing of context services. The problem is obviously a multi-objective optimization problem (MOO), where it might be possible only to find a semi-optimal solution as requests can be contradicting. For example, one request can query for the cheapest provider whereas another queries for the most accurate but also more expansive one. As a result, one or both of these requests have to cut back. In general and in contrast to a single-objective optimization, a single solution for such a MOO problem does not exist. It is rather a set of optima. These

#	o1	o2	o3	o4	$\sum cost$	$u_c$	$u_{q1}$	$u_{q2}$	$u_{q3}$	AUF
1					0	1	-	-	-	0
2				✓	30	0.84	-	-	0.87	0
3			✓		5	0.97	0.00	0.49	-	0
4			✓	✓	35	0.81	0.00	0.41	0.86	1.04
5		✓			50	0.73	0.80	0.76	-	0
6		✓		✓	80	0.57	0.80	0.68	0.73	1.39
7		✓	✓		55	0.70	0.80	0.75	-	0
8		✓	✓	✓	85	0.54	0.80	0.67	0.72	1.37
9	✓				100	0.460	1.00	0.73	0.23	1.21
10	✓			✓	130	0.30	1.00	0.65	0.60	1.27
11	✓		✓		105	0.43	1.00	0.72	0.22	1.18
12	✓		✓	✓	135	0.27	1.00	0.63	0.59	1.24
13	✓	✓			150	0.19	1.00	0.59	0.09	0.94
14	✓	✓		✓	180	0.03	1.00	0.51	0.46	1.00
15	✓	✓	✓		155	0.16	1.00	0.58	0.08	0.91
16	✓	✓	✓	✓	185	0	1.00	0.50	0.45	0.97

**Table 9.6:** Selection Example 3 – Results

optimal solutions are also called Pareto optimal solutions [25, 94]. Such a set contains solutions that cannot be improved in one of the single objectives without deteriorating their performance in at least one of the other objectives.

Marler et al. discuss in detail different methods for MOO with the main result that “the selection of a specific method depends on the type of information that is provided in the problem, the user’s preference, the solution requirements, and the availability of software” [89, 90]. According to Marler et al., MOO methods can be generally divided into methods with a priori articulation of preferences (e.g. like the weighted sum method used in this work) and methods with a posteriori articulation of preferences (e.g. physical programming) [90]. Additionally, they mention methods with no articulation of preferences (like the global criterion methods) and genetic algorithms [89].

A MOO problem can have several different solutions. Consequently, a decision maker has to select one of the solutions from this set of possible solutions as the most preferable solution. For using methods with a priori articulation of preferences, the decision maker has to express preferences or additional criteria before calling the method. This has the advantage that these methods return only one solution. However, the decision maker has to express his preferences without knowing different potential solutions. In contrast, methods with a posteriori articulation of preferences return the Pareto optimal set. This has the advantage that the decision maker knows the different alternatives. But the selection process is then up to the decision maker.

In our work, we actually do not have a single decision maker, but rather several context consumers that request different context information. Furthermore we have the system



requirement to minimize the total resource consumption. Methods returning a set of potential solutions are not sufficient in this work, as these would require an additional selection process. We consequently selected a method with a priori articulation of preferences, namely the weighted sum method. According to Marler et al., “the weighted sum method provides a basic and easy-to-use approach for multi-objective optimization [...]” [90].

Disregarding these benefits of the weighted sum method, this method also has some disadvantages. One of the biggest issues is the selection of the weights. Even if several approaches exist to systematically determine the weights (see e.g. the survey by Eckenrode [37]), it cannot be guaranteed that the solution is optimal in the end. In addition, it is not possible to evaluate points on non-convex partitions of the Pareto optimal set in the search space, as mentioned by Marler et al. [89]. However, Das et al. stated that non-convex Pareto optimal sets are relatively uncommon [31]. Further drawbacks are that the relations between weights and result are often complicated and non-intuitive and the uneven sampling of the Pareto front. In addition, the method requires a scaling of the different parameters (which is no problem in our case as we already support representations).

The AUF and the brute-force search for a set of offers respectively chains with the highest result for that AUF is only a simple solution with some drawbacks, e.g. with regard to scalability. Even if we increased the complexity of our selection algorithm from  $2^{|O_M|}$  to  $(|O_{CS}| + (|CR| \cdot |O_S|)^{|R_{CR}|})^{|R_{CS}|}$ , it would still scale exponentially with the number of queries. The scalability is also discussed from the practical point of view in the evaluation chapter (see Chapter 12). Nevertheless, the focus was to support simple and independently specifiable context request and not that much on high scalability. To further increase scalability, also other heuristic methods like Evolutionary Algorithms (see [147]), A\*-search [50] or deterministic optimisation methods may be evaluated in future work.

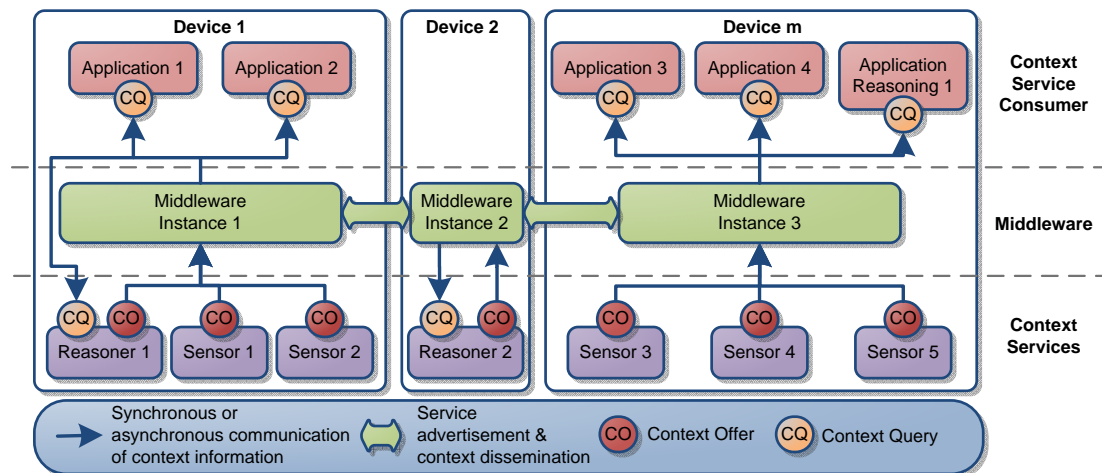


# 10 Architecture

*Science bestowed immense new powers on man and at the same time created conditions which were largely beyond his comprehension and still more beyond his control.*

– Winston Churchill (1874-1965)  
Speech, 31st March 1949.

Based on the concepts presented in the previous chapters, a context management middleware has been developed. Figure 10.1 presents an overview of its general infrastructure and execution environment. As depicted, a middleware instance has to run on every device on which at least one context consumer or service is executed<sup>1</sup>. This instance can be shared by all context consumers and services on that device. Context consumers register requests on that instance and context services register respectively offers. The middleware instance then organizes the selection and also handles the advertisements of context services regarding external context consumers.



**Figure 10.1:** Context Service Infrastructure and Execution Environment

Also the context dissemination between a context service and a context consumer, which is placed on another device as the service, is handled via middleware instances of the service and the consumer. Even if this causes a small overhead compared to a direct communication of the context service with the middleware instance of the consumer

<sup>1</sup>It is also possible to implement stand-alone context services that do not require any middleware instance by providing these services the remote binding and advertisement functionalities.

requested for the information<sup>2</sup>, this has the benefit that context services do not have to provide additional techniques for service advertisement and for communication.

Figure 10.2 presents the architecture of the middleware. The middleware is split into several separate components. For that reason, the middleware is easily extendable by e.g. exchanging one of the components. This has also been a requirement to further ease the future work on e.g. other matching algorithms<sup>3</sup> or other selection processes<sup>4</sup>. Last but not least, it is a requirement to add more powerful adaptation mechanisms regarding the middleware itself. In Section 2.3 also the different adaptation mechanisms have been introduced. Our middleware is a composition of the different components. As the implementation already foresees different variants of the specific components (e.g. more than one implementation of the context repository), this is one step towards a self-adaptive middleware. With a self-adaptive middleware, also the middleware adapts to context changes, e.g. a less resource-consuming selection algorithm could be chosen if the device is running out of battery. Additional to the component-based adaptation, the middleware could also be adjusted by tuning the different parameters (e.g. the parameter  $\alpha$  introduced in Section 9.2.2). The actual adaptation of the middleware is still future work.

The system design in general is motivated by the different processes described in the previous chapters (mainly the matching approach described in Chapter 8 and the selection approach described in Chapter 9). The general process model (see Figure 1.1) can be summarized as followed:

1. *Context Services* and *Context Service Consumers* register their offers and queries, respectively at the *Discovery Service*.
2. These offers and queries have to be matched, as described in Chapter 8. For this purpose, the component called *Matching Service* is used.
3. After the matching, one of the matching offers has to be selected as context provider. This is done by the *Selection Service*.
4. Finally the *Binding Service* is used to establish the binding between the selected context service and the consumer.

Additionally to the components motivated by this general process model, we have introduced several other components, which extend these basic functionalities:

- The *Ontology Manager* encapsulates the access to the context ontology (see Chapter 6).
- The *Mediation Engine* provides the techniques to establish the mediator chains as described in Section 8.2. Even if this is actually a part of the matching phase, we decided to split matching (step 1 and 3 of the matching phase described in Chapter 8) and mediation (step 2 of the matching phase) mainly to ease the further independent development of both functionalities.

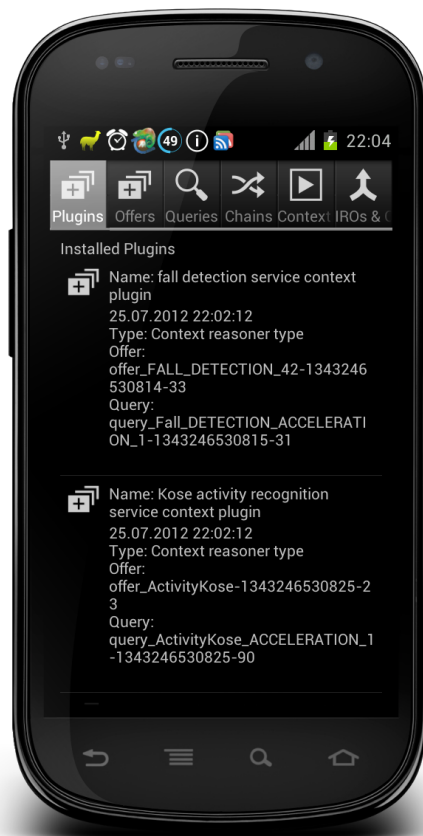
---

<sup>2</sup>Between local service and consumer, it is also required that the exchange is between service and middleware and middleware and consumer. One reason is that the middleware provides the mediation techniques. Another reason is the monitoring and filtering done by the middleware which is essential to react e.g. on services providing information which violate its context offer.

<sup>3</sup>See Section 8.5 for more details.

<sup>4</sup>See Section 9.6 for more details.





**Figure 10.3:** Screenshot of the Android Visualization Application

the COQL as introduced in Chapter 7) at the *Discovery Service*. To discover not only local context services, the *Discovery Service* can leverage several *Discovery Plugins*. These plugins provide support for different remote discovery protocols, e.g. a protocol using ZooKeeper [60] as a remote service dictionary. The *Discovery Service* employs a local repository containing all registered context offers and queries. For every new registration of either a query or offer, the *Discovery Service* triggers a new matching process by the *Matching Service*. For calculating the *Matching Results* as described in Chapter 8, the *Matching Service* accesses the ontology with the help of the *Ontology Manager*, triggers the mediation check (as described in Section 8.2) by the *Mediation Engine* and accesses the repository for context offers and queries provided by the *Discovery Service*.

The *Mediation Engine* encapsulates the establishment of the mediator chains as described in Section 8.2. The resulting chain offers are first returned to the *Matching Service*, which then proceed to the constraints matching (see Section 8.3). If the metadata constraint matching is also successful, the established mediator chains will be stored in the *Mediator Chain Repository* of the *Binding Service*. If mediator chains binding a request of a context reasoner are established, this is also reported to the *Discovery Service*, which forwards the offer of the respective reasoner if all queries of that reasoner are resolved. In general,

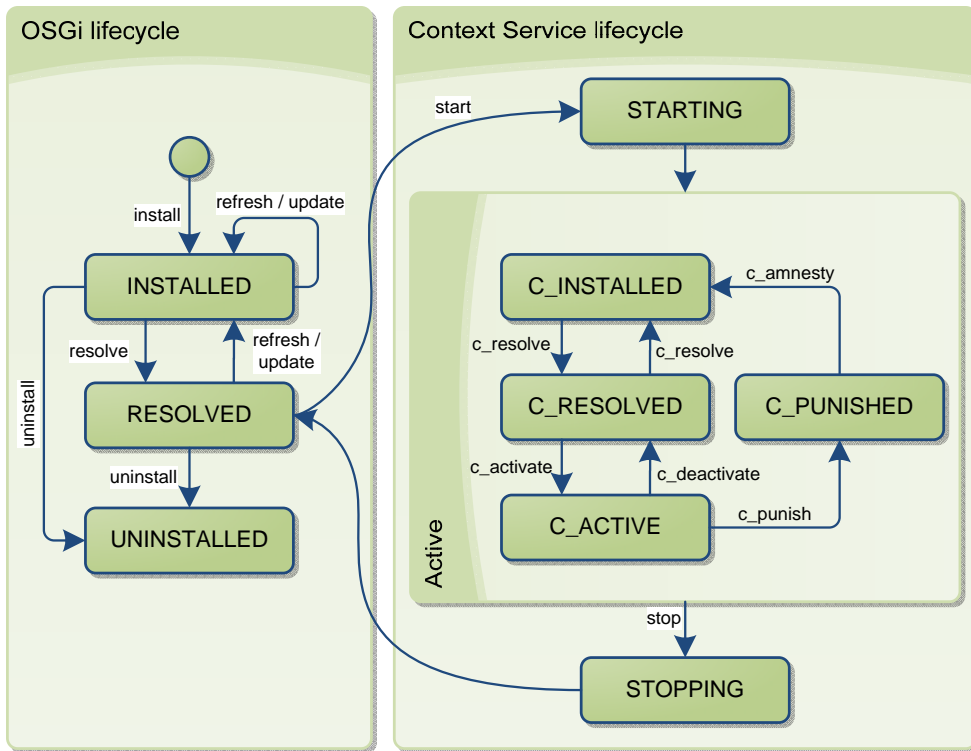
the *Binding Service* also reports chains to the *Selection Service*. Here it has to be differentiated between chains for a query of a reasoner and chains for other queries. Chains for a query of a reasoner are only sent to the *Selection Service* if at least one chain exists that is using the respective context reasoner. Otherwise these chains are ignored as they are not required. However they have to be established even before the usage of the according reasoner is clear to ensure that the queries of the reasoner can be resolved.

If both event queues of the *Discovery Service* and the *Matching Service* are empty, a new selection process done by the *Selection Service* is triggered (see Chapter 9 for a detailed description of the selection process). The *Selection Service* accesses the *Mediator Chain Repository* and the *Context Offer & Query Repository* provided by the *Discovery Service*. After selecting a mediator chain for a certain context query, the *Selection Service* triggers the activation of the mediator chain, which consequently binds the offer and query used in the chain by the *Binding Service*. More precisely, the *Binding Service* stores the selected mediator chains created by the mediation check during the matching phase and calls the according chains for an offer if this offer provides new information. The *Binding Service* also sends the command to activate or deactivate a selected respectively unselected context offer. The *Discovery Service* forwards this command to the respective context service. To be able to also access/receive information from remote context services, the *Binding Service* as well can use several *Binding Plugin* for e.g. establishing a Socket based connection between two middleware instances.

This *Binding Service* uses the mediator chains, which have been established during the matching phase by the *Mediation Engine*. Furthermore it triggers the monitoring process by the *QoC and CoC Monitor*. The monitor controls the context data provided by the context provider according to their context offers and handles eventual violations of the constraints, e.g. a new matching process can triggered whereby the violating offer is ignored.

The prototype implementation of the middleware is based on Java and OSGi [102]. The different components described in the previous section on the architecture are realized as OSGi bundles. Similar to Paspallis [104], also context providers are implemented leveraging the OSGi component specification. We have extended the general *OSGi lifecycle* as the context services can have additional context dependencies. The context providers for these dependencies change dynamically as new services appear or disappear and as new applications are started or stopped.

To overcome this limitation, the *Context Service lifecycle* is amended with four context middleware specific states as depicted in Figure 10.4: C\_INSTALLED, C\_RESOLVED, C\_ACTIVE and C\_PUNISHED. A newly installed context service is registered as C\_INSTALLED. If all context dependencies are resolved, this service is moved to C\_RESOLVED. Context services (e.g. sensors) without additional context dependencies are directly moved to C\_RESOLVED after the registration. For context reasoners, a matching process is triggered for all of its context queries. Only if these matching processes result in nonempty matching results, the reasoner is moved to C\_RESOLVED. This extended context service lifecycle has already been introduced by Paspallis [104]. We extended this lifecycle with the additional state C\_PUNISHED. A running context service is monitored by the *QoC and CoC Monitor* and offenses against the constraints in its offer, will be punished: context services are moved to the state C\_PUNISHED and will be ignored. For all components of the middleware except the *Discovery Service*, who



**Figure 10.4:** Context Service Lifecycle

manages the context service lifecycles, and the *QoC and CoC Monitor*, punished services are not existent. In the current implementation, context services are moved after a timeout<sup>5</sup> of  $x$  minutes back into the C\_INSTALLED state and will then be used like any other service. For future work it is also foreseen to have additional amnesty conditions monitored by the *QoC and CoC Monitor*. For example, a service providing a position, which is moved to C\_PUNISHED as it doesn't hold its constraint regarding the provided accuracy, can be amnestied after moving a certain amount from the position where it violates the constraint.

<sup>5</sup>This timeout is also a system parameter and configurable.



**Part III**  
**Evaluation**



# 11 Demonstrators

---

*I have not been able to discover the cause of those properties of gravity from phenomena, and I frame no hypotheses; for whatever is not deduced from the phenomena is to be called a hypothesis, and hypotheses, whether metaphysical or physical, whether of occult qualities or mechanical, have no place in experimental philosophy.*

– Isaac Newton (1642-1726)

Letter to Robert Hooke (15th February 1676)

In this chapter, we present several demonstrators based on real-world scenarios. For this purpose, a large amount of existing works has been reviewed to collect which kind of context information including its metadata are used. This study serves as the basis for the different demonstrators. More precisely the context sources presented in the different existing works are implemented as context services. A subset (in the Demonstrators C–E also the complete set) of these context services is used with the different demonstrators.

As described in Chapter 5 our system interacts with a set of context services and a set of context consumers. These consumers may be different context-aware applications or small parts of such applications used to receive a certain kind of context information. In order to demonstrate the usefulness of the explained approach and the fulfilment of the requirements introduced in Section 1.2, we do not require fully implemented context-aware systems but rather have to implement the context consuming parts of these applications. Hence, a demonstrator consists of a set of context services, a set of Inter-Representation Operations (IROs) and metadata operators, and one or more context consumers. Every context consumer registers a context query at the system. With the different demonstrators we focus on the different requirements presented in Section 1.2. The first Demonstrator A presents different examples for addressing the issue of heterogeneous represented context information (see Requirement 3). In Demonstrators B and C the dynamic matching and selection of context offers are discussed (see Requirement 6). The next Demonstrator D shows an example for the remote discovery and binding of context information (see Requirement 1). Within this demonstration, the issues of loose coupling (see Requirement 2) and the provisioning of context offers / requests (see Requirement 4) are also discussed implicitly. Finally the fifth Demonstrator E focuses on the cost minimization by explicit activation respectively deactivation of required and not required context services (see Requirement 7 and Requirement 5).

## 11.1 Study of Context Information, QoC, and CoC in Related Work

In order to prepare the demonstration of our work and to build the demonstration on a solid and realistic scenario, we have studied existing works regarding the types of context information, QoC, and CoC, that are described in related works (also in the examples). This study is not exhaustive but is rather used as input for developing a realistic set of context services.

**Android:** The Android system already supports several different context providers as depicted in Table 11.1, but for these providers, the system does not offer any quality metrics. Additionally to the depicted sensors, the system provides different location sources<sup>1</sup>, namely GPS, Cell-ID, and WiFi. These location providers also calculate the accuracy of the retrieved position as a radius in meter. Furthermore, the providers are coarsely divided regarding their power consumption into POWER\_HIGH (GPS sensor), POWER\_LOW (Cell-ID- based position), and POWER\_MEDIUM (WiFi based position). When registering a listener for a specific sensor, it is necessary to consider a sampling rate. Unfortunately the Android API does not support the specification of concrete sampling rates but rather of sampling rate classes, namely SENSOR\_DELAY\_NORMAL, SENSOR\_DELAY\_UI, SENSOR\_DELAY\_GAME, or SENSOR\_DELAY\_FASTEST.

Sensor	Type	Description
TYPE_ACCELEROMETER	Hardware	Acceleration force in $m/s^2$ that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.
TYPE_AMBIENT_TEMPERATURE	Hardware	Ambient room temperature in degrees Celsius ( $^{\circ}C$ ).
TYPE_GRAVITY	Software or Hardware	Force of gravity in $m/s^2$ that is applied to a device on all three physical axes (x, y, and z).
TYPE_GYROSCOPE	Hardware	A device's rate of rotation in $rad/s$ around each of the three physical axes (x, y, and z).
TYPE_LIGHT	Hardware	Ambient light level (illumination) in $lux$ .
TYPE_LINEAR_ACCELERATION	Software or Hardware	Acceleration force in $m/s^2$ that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.
TYPE_MAGNETIC_FIELD	Hardware	Ambient geomagnetic field for all three physical axes (x, y, and z) in $\mu T$ .
TYPE_ORIENTATION	Software	Degrees of rotation that a device makes around all three physical axes (x, y, and z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the <code>getRotationMatrix()</code> method.
TYPE_PRESSURE	Hardware	Ambient air pressure in $hPa$ or $mbar$ .

<sup>1</sup>See <http://developer.android.com/guide/topics/location/obtaining-user-location.html> for more details.

Sensor	Type	Description
TYPE_PROXIMITY	Hardware	Proximity of an object in <i>cm</i> relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.
TYPE_RELATIVE_HUMIDITY	Hardware	Relative ambient humidity in percent.
TYPE_ROTATION_VECTOR	Software or Hardware	Orientation of a device by providing the three elements of the device's rotation vector.
TYPE_TEMPERATURE	Hardware	Temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14

**Table 11.1:** Sensor Types Supported by the Android Platform.<sup>2</sup>

Paller has evaluated these sampling classes for the accelerometer on a Google Nexus One smartphone<sup>3</sup> (SENSOR\_DELAY\_NORMAL results in an average frequency of 4.09 Hz, SENSOR\_DELAY\_UI in 9.87 Hz, SENSOR\_DELAY\_GAME in 16.16 Hz, or SENSOR\_DELAY\_FASTEST in 24.45 Hz) [103]. On a Sony-Ericsson x10 mini SENSOR\_DELAY\_NORMAL results in an average frequency of 4.74 Hz, SENSOR\_DELAY\_UI in 14.15 Hz, SENSOR\_DELAY\_GAME in 32.55 Hz, or SENSOR\_DELAY\_FASTEST in 94.77 Hz. Similarly he has evaluated the battery usage of the accelerometer on the Google Nexus One depending on the chosen frequency. The results are depicted in Table 11.2.

Sampling speed	Battery consumption (%/hour)
Normal	1.71
UI	3.19
Game	3.27
Fastest	3.41

**Table 11.2:** Battery Consumption of the Accelerometer on a Google Nexus One

**Agostini et al.:** Agostini et al. give an example on how to infer the music preferences of a user based on his list of artists. But they do not use QoC or CoC information [4].

**Dai et al.:** Dai et al. describe an approach for retrieving information about user falling based on the acceleration of the mobile [30, 29]. They do not provide any quality related information. Nevertheless they evaluated their approach: “[...] the average FN [False Negative] value is 2.67%, while the FP [False Positive] value is 8.7%.” Furthermore, they also evaluated the average resource consumption: “The average CPU usage is 7.41%; the memory usage is about 600KB, 0.6% of total RAM capacity of G1 phone”<sup>4</sup>. Furthermore,

<sup>2</sup>Copied from [http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html).

<sup>3</sup>Google Nexus One [http://en.wikipedia.org/wiki/Nexus\\_One](http://en.wikipedia.org/wiki/Nexus_One).

<sup>4</sup>HTC Dream also known as T-Mobile G1, 528 MHz, 192 MB RAM [http://en.wikipedia.org/wiki/HTC\\_Dream](http://en.wikipedia.org/wiki/HTC_Dream)

they extended the reasoning approach by also using the magnetic field sensor of the mobile and a magnetic accessory bound on the leg of the user. With this approach they increased the accuracy: “[...] the average of FN is 2.13% and the FP value is 7.7%”. Even if they do not provide QoC values dedicated for every context information, these values can at least be used to roughly describe the two approaches.

**Rouvoy et al.:** Rouvoy et al. introduce a detailed example on retrieving hierarchically high-level context information from low-level data [113].

- Some of the low-level information are:
  - RFID tag presence
  - preference of the user regarding advertisements (e.g. no advertisements, only postal advertisements or advertisements via email)
  - battery change state
  - battery time left
  - bluetooth link quality
  - WiFi link quality
  - WiFi bit rate
- Some of the high-level information:
  - Bluetooth notification enabled
  - WiFi browsing enabled
  - WiFi download enabled
- The interconnection and also the intermediate context information are described by Rouvoy et al. [113]. Abid et al. reuse the example of Rouvoy et al. and added QoC data [1]:
  - Trustworthiness
  - Up-to-dateness
  - Precision

**Kwapisz et al.:** Kwapisz et al. present an activity recognition approach, which retrieves the activity of the user only based on data from the accelerometer [76]. Their approach relies on supervised learning methods (The authors compared different learning methods like J48 decision tree and Multilayer Perceptron). They considered six activities: walking, jogging, ascending stairs, descending stairs, sitting, and standing. Their approach recognizes the activity correctly over 90% of the time (accuracy is 61.5 % correctly predicted activities in minimum and 98.3 % in maximum depending of the actual activity). Based on the results in the paper, an Android application called Actitracker<sup>5</sup> has been developed.

**Lee et al.:** Similar to Kwapisz et al., Lee et al. also developed an activity recognition approach utilizing the accelerometer [81]. In difference to Kwapisz et al., Lee et al. use an hierarchical hidden markov model and are able to recognize the activities *standing*, *walking*, *running*, and *stair up/down*. Unfortunately the authors do not provide precise QoC values.

---

<sup>5</sup><http://www.actitracker.com/>

**EEMSS:** As described in Section 4.1.9, Wang et al. demonstrated their approach called *Energy Efficient Mobile Sensing System (EEMSS)* by developing an activity recognition system retrieving the users current activity based on data from GPS, accelerometer, microphone and a WiFi detector [144]. Unlike other approaches, the four used sensors are not running permanently but rather are activated only when they are useful to detect an activity change. They have also precisely measured the energy consumption of the different sensors on a Nokia N95 smartphone, e.g. the accelerometer with a duty cycle of 6 sec sensing and 10 sec sleeping requires 0.359 Joule per sample. Furthermore, “the average recognition accuracy [...] is found to be 92.56% with a standard deviation of 2.53%” [144]. The authors have also evaluated how long a state transition and hence the detection of a new activity lasts. The results are summarized in Table 11.3.

	Walking	Vehicle	At some place
Walking	–	< 40 sec	< 5 min
Vehicle	< 1.5 min	–	–
At some place	< 1 min	–	–

**Table 11.3:** Activity Detection Time in the Activity Recognition Approach by Wang et al. [144]

Regarding the total resource consumption of their activity recognition system, the authors only mention that “[...] the average device lifetime result with EEMSS [Energy Efficient Mobile Sensing System] application running on a fully charged Nokia N95 device is 11.33 hours with regular cell phone functionalities” [144]. However a reference value regarding the average runtime of the phone is missing.

**Kose et al.:** Kose et al. evaluated different classification methods for online activity recognition on smartphones using the accelerometer [74]. They focused on the activities *walking*, *running*, *sitting*, and *standing*. It turned out that the *clustered KNN* algorithm, which is an improvement of minimum distance and k-nearest neighbor (KNN) classification algorithms, provides the highest accuracy (92.12%) for all activities. Additionally to the measurement of the accuracy of the different classification algorithms, the authors evaluated the resource consumption as well, e.g. the clustered KNN causes 29% CPU usage and 21.9 MB memory usage on a Samsung Galaxy Gio<sup>6</sup>.

**Yan et al.:** Another activity recognition approach utilizing the accelerometer is the system by Yan et al. [149]. In difference to the previous works, they recognize 10 different activities, namely *stand*, *slowWalk*, *sitRelax*, *sit*, *normalWalk*, *escalatorUp*, *escalatorDown*, *elevatorUp*, *elevatorDown*, and *downStairs*. They developed an approach called *adaptive accelerometer activity recognition (A3R)* based on the J48 adaptive decision tree classifier of the Weka toolkit<sup>7</sup>, which dynamically adapts the sampling frequency of the accelerometer in order to save energy. They evaluated the accuracy and power consumption for different combinations of sampling frequencies and types of computed features. This evaluation results in the configuration depicted in Table 11.4.

<sup>6</sup>Samsung Galaxy Gio, 800 MHz processor, 278 MB RAM, [http://www.gsmarena.com/samsung\\_galaxy\\_gio\\_s5660-3741.php](http://www.gsmarena.com/samsung_galaxy_gio_s5660-3741.php).

<sup>7</sup>“Weka is a collection of machine learning algorithms for data mining tasks” <http://www.cs.waikato.ac.nz/ml/weka/>.

Activity	Frequency	Computed features	Accuracy	Power consumption (J/hr)
stand	16 Hz	$F_{time}$	0.9516	79.95
slowWalk	16 Hz	$F_{time}$	0.9171	79.95
sitRelax	5 Hz	$F_{time} + F_{freq}$	0.9823	75.8
sit	16 Hz	$F_{time}$	0.9855	79.95
normalWalk	16 Hz	$F_{time}$	0.9237	79.95
escalatorUp	50 Hz	$F_{time}$	0.7265	110.05
escalatorDown	100 Hz	$F_{time} + F_{freq}$	0.756	230.75
elevatorUp	5 Hz	$F_{time}$	0.7827	55.35
elevatorDown	5 Hz	$F_{time}$	0.8056	55.35
downStairs	16 Hz	$F_{time}$	0.8344	79.95
average			0.86654	92.705

**Table 11.4:** Evaluation of Accuracy and Power Consumption in the Activity Recognition Approach by Yan et al. [149]

Yan et al. give a detailed explanation of the computed features [149]. The average values depicted in the last row expect an equal distribution of all activities. These values are not provided by Yan et al. and are calculated by the author of this thesis. Instead Yan et al. made an extensive user study to retrieve the distribution of the different activities but they only presented aggregated and raw results of this study. These results are not useful for a context description within a context offer.

## 11.2 General Description of the Demonstrators

As described in Chapter 5 our system interacts with a set of context services (context providers) and a set of context consumers. These consumers need not be different context-aware applications but can be small parts of such an application used to receive a certain kind of context information. Thus in order to demonstrate the usefulness of the explained approach and the fulfilment of the requirements introduced in Section 1.2, we do not require fully implemented context-aware systems but rather have to implement the context consuming parts of these applications. These parts are called context listeners. In summary, a demonstrator consists of our context middleware, a set of context services, one or more context listeners, a set of Inter-Representation Operations, and a set of metadata operations. In parallel to this, the visualization application is running (see Chapter 10 for a description of the visualization application). Screenshots of this application are used to illustrate the different scenarios.

Based on the previous study on context types, QoC and CoC referenced in the literature, we have implemented a wide range of context services. Table 11.5 summarizes the



context offers of these services<sup>8</sup>. Nearly all implemented context services in our demonstrators are fully working and provide the context information as specified in the context offers. Only these context services and context offers marked with asterisk (\*) in the table are not fully implemented. These services are rather implemented as mockups able to encapsulate the respective context retrieving mechanisms. However, even these mockups randomly fire context information according to the respective offer.

Service	Context Offer	Entity	Offered Scope	Representation	Metadata Constraints
Acceleration	Google Nexus One SENSOR_DELAY_NORMAL	Device   this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)	TimeStamp $\geq 1$ UnixTimeStamp $\wedge$ PowerConsumption = 1.71 PercentPerHour $\wedge$ ChangeFrequency = 4.09 Hz
	Google Nexus One SENSOR_DELAY_FASTEST	Device   this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)	TimeStamp $\geq 1$ Unix-TimeStamp $\wedge$ Sensor-Cost = 3.41 Percent-PerHour $\wedge$ Change-Frequency = 24.45 Hz
	Sony Ericsson X10 mini SENSOR_DELAY_NORMAL	Device   this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)	TimeStamp $\geq 1$ Unix-TimeStamp $\wedge$ Change-Frequency = 4.74 Hz
	Sony Ericsson X10 mini SENSOR_DELAY_FASTEST	Device   this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)	TimeStamp $\geq 1$ Unix-TimeStamp $\wedge$ Change-Frequency = 94.77 Hz
	Other devices SENSOR_DELAY_NORMAL	Device   this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)	TimeStamp $\geq 1$ UnixTimeStamp $\wedge$ ChangeFrequency $> 4$ Hz $\wedge$ ChangeFrequency $< 10$ Hz
	Other devices SENSOR_DELAY_FASTEST	Device   this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)	TimeStamp $\geq 1$ Unix-TimeStamp $\wedge$ Change-Frequency $> 20$ Hz $\wedge$ ChangeFrequency $< 100$ Hz

<sup>8</sup>We simplified the table by removing the context service label if this service only provides one context offer, e.g. the context service *GPS* only provides the offer *GPS*.

Service	Context Offer	Entity	Offered Scope	Representation	Metadata Constraints
	GPS	Device   this	Location	LocationWGS84 (Latitude, Longitude, Altitude)	Accuracy < 50 meter ^ TimeStamp ≥ 1 UnixTimeStamp ^ EstimatedSensorCost = 100
	Light	Device   this	AmbientLight-Level	Float_lux	TimeStamp ≥ 1 UnixTimeStamp ^ EstimatedSensorCost = 5
	Magnetic field	Device   this	MagneticField	AndroidMagneticFieldRep (X_MagneticField, Y_MagneticField, Z_MagneticField)	TimeStamp ≥ 1 UnixTimeStamp ^ EstimatedSensorCost = 5
	Network location	Device   this	Location	LocationWGS84 (Latitude, Longitude, Altitude)	Accuracy < 1000 meter ^ TimeStamp ≥ 1 UnixTimeStamp ^ EstimatedSensorCost = 50
	Orientation	Device   this	Orientation	Android-OrientationRep (Azimuth, Pitch, Roll)	TimeStamp ≥ 1 UnixTimeStamp ^ EstimatedSensorCost = 5
	Pressure	Device   this	Atmospheric-Pressure	Float_hPa	TimeStamp ≥ 1 UnixTimeStamp ^ EstimatedSensorCost = 5
	Proximity	Device   this	Proximity	Float_cm	TimeStamp ≥ 1 UnixTimeStamp ^ Sensor-Cost = 5
	Rotation	Device   this	RotationRate	Android-RotationRateRep (X_RotationRate, Y_RotationRate, Z_RotationRate)	TimeStamp ≥ 1 UnixTimeStamp ^ EstimatedSensorCost = 5
	Fall detection (*)	Device   this	FallDetection	BooleanRep	FalseNegativeRate = 2.67% ^ FalsePositiveRate = 8.7% ^ DeviceMemoryUsage = 600 kB ^ CPUUsage = 3.128 MHz
	Music preferences (*)	User   this	Music-Preferences	StringRep	

Service	Context Offer	Entity	Offered Scope	Representation	Metadata Constraints
	Advertisement preference (*)	User   this	Advertisement-Preference	StringRep	
Battery	Battery charging state	Device   this	BatteryAC-PowerStatus	StringRep (ACOffline, ACOOnline)	
	Battery level	Device   this	BatteryLoad	PercentRep	
	Bluetooth	Device   this	Discovered-Bluetooth-Device	String	
WiFi	WiFi Strength	Device   this	Network-SignalStrength	Float_Percent	
	WiFi Networks	Device   this	Discovered-WifiNetwork	String	
	Activity recognition 1 (*)	User   this	Activity	Activity-KwapiscRep: String (walking, jogging, ascendingStairs, descendingStairs, sitting, standing)	Accuracy > 61.5% ^ Accuracy < 98.3%
	Activity recognition 2 (*)	User   this	Activity	ActivityLeeRep: String (standing, walking, running, stairUpDown)	
	Activity recognition 3 (*)	User   this	Activity	ActivityWangRep: String (working, meeting, officeLoud, resting, homeTalking, homeEntertaining, placeQuiet, placeSpeech, placeLoud, walking, vehicle)	Accuracy = 92.56% (2.53% Delta) ^ RecognitionTime < 5 min
	Microphone	Device   this	MicrophoneLog	Adaptive Multi-Rate audio codec (AMR_NB)	
	Activity recognition 4 (*)	User   this	Activity	ActivityKoseRep: String (walking, sitting, running, standing)	Accuracy = 92.12% ^ DeviceMemoryUsage = 21.9 MB ^ CPUUsage = 232 MHz

Service	Context Offer	Entity	Offered Scope	Representation	Metadata Constraints
	Activity recognition 5 (*)	User this	Activity	ActivityYanRep: String (stand, slowWalk, sitRelax, sit, normalWalk, escalatorUp, escalatorDown, elevatorUp, elevatorDown, downStairs)	Accuracy = 86.654% ^ PowerConsumption = 92.705 Joule/Hour
Calendar	Calendar Activity	User this	Activity	Activity-CalendarRep: String (free, busy, ...)	Cost = 0
	Calendar Events	User this	Activity	EventDescription (EventTitle, StartTime, EndTime, Attendees)	Cost = 0
	Contacts	User this	KnownPersons	PersonDescription (Name, Email, ...)	Cost = 0

**Table 11.5:** Context Offers of the Implemented Context Services

Table 11.5 describes the different context services with the respective offers. A context service can provide more than one context offer. For example, the context service *Acceleration* provides 6 different context offers (Google Nexus One SENSOR\_DELAY\_NORMAL, Google Nexus One SENSOR\_DELAY\_FAST, ...). At runtime this service only provides two offers depending on the device (e.g. offer 1 and 2 are only applicable on the device *Google Nexus One*). For every offer the characterized context entity type, the scope, the representation<sup>9</sup> and the different metadata constraints are shown.

As already indicated in the previous section, several of the context service contain reasoning mechanisms retrieving new context information from other (basically) low-level information. These context services are all different services encapsulating the activity reasoners and also the service for fall detection. The information the different services require are described in Table 11.6. Similar to the table on context services and its offers, the different context requests of the context services are fully described. For example, the context service *Activity recognition 3* requires the information regarding the four context scopes *Acceleration*, *Location*, *DiscoveredNetworks*, and *MicrophoneLog*. The information provider for the *Location* scope has to fulfil the requirement that the accuracy of the location is less than 500 meter. Additionally, within the selection function of this request a preference for minimizing the accuracy is expressed<sup>10</sup>.

<sup>9</sup>For composite representations, the different dimensions are depicted in brackets after the label of the composite representation.

<sup>10</sup>From the general understanding of accuracy it is preferable to maximize the accuracy. But in our case,

Context Consumer	Entity	Required Scope	Representation	Metadata Constraints	Selection Criteria
Fall detection	Device this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)		min ChangeFrequency
Activity recognition 1	Device this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)		
Activity recognition 2	Device this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)		
Activity recognition 3	Device this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)		
	Device this	Location	LocationWGS84 (Latitude, Longitude, Altitude)	Accuracy < 500 meter	min Accuracy
	Device this	Discovered-WifiNetwork	String		
	Device this	Microphone-Log	AMR_NB		
Activity recognition 4	Device this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)		
Activity recognition 5	Device this	Acceleration	AndroidDevice-AccelerationRep (x_Acceleration, y_Acceleration, z_Acceleration)	Change-Frequency > 10 Hz	min ChangeFrequency

**Table 11.6:** Context Queries of the Implemented Context Services

also the representation of the accuracy has to be taken into account: in this example accuracy is described as a radius in meter. Consequently the smaller the radius, the larger the actual accuracy is. We adopted this description and representation from the Android system.

## 11.3 Demonstrator A: Heterogeneity

In this first Demonstrator A we focus on the requirement to handle heterogeneously represented context information (see Requirement 3). In order to illustrate all facets of this aspect, we present two different scenarios in this first demonstration. While the demonstrator consists of the previously introduced context services, the middleware, and the visualization application running on the client, the scenarios differ in the context listener and the corresponding context request. A context listener is a small part within a context-aware application that is responsible for querying and receiving context information from the middleware. In the two presented scenarios, only one context listener registers its query at the middleware. The corresponding context queries of both scenarios are shown in Table 11.7. The scope, representation, metadata constraints, and the selection criteria are depicted similar to the context requests of the context services (see Table 11.6). Here, we request information for a concrete entity (here: *mwagner@vs.uni-kassel.de*) in contrast to the previous queries which requested information regarding entity types<sup>11</sup>.

In the first scenario, the context listener is querying for the location of the user *mwagner@vs.uni-kassel.de*, whereby the location should be represented as an address. The selected context provider should offer the requested location with a freshness less than 10 seconds. For the selection of the context provider, the context listener sets a high preference (precisely 0.9) on using a provider that minimizes the cost (here: *EstimatedSensorCost*) and only a low preference (namely 0.1) on minimizing the accuracy<sup>12</sup>.

In the second scenario, the listener requests the current activity of the same user. In difference to the first scenario, the request does not contain metadata constraints or selection criteria.

Context Consumer	Entity	Required scope	Representation	Metadata constraints	Selection criteria
Scenario 1	User   mwagner@ vs.uni- kassel.de	Location	AddressRep (Street, Number, City, Country)	Freshness < 10 s	0.9 min Estimated- SensorCost & 0.1 min Accuracy (m)
Scenario 2	User   mwagner@ vs.uni- kassel.de	Activity	ActivityKoseRep: String (walking, sitting, running, standing)		

**Table 11.7:** Demonstrator A: Context Queries

Both scenarios of this demonstrator mainly focus on the first two steps of the context offer and query matching presented in Chapter 8: the initial matching and the mediation

<sup>11</sup>Nevertheless, these entity types are concreted at runtime by replacing them with the concrete entity. More precisely the place holder *this* is replaced with the id of the device respectively the user ID.

<sup>12</sup>The accuracy is represented as a radius in meter. Hence, it is preferable to minimize it.

check. The initial matching only coarsely compares an offer from the set of all offers with a query in order to find offers which are potentially useful as context provider for the query. For this initial matching test basically the scope and the entity are compared. The result of this first check is depicted in Table 11.8. In the column ‘potential context offers’, the context services are collected offering the correct scope and entity as requested in the respective scenario. The offered representation of the respective context offer is shown in brackets after the name of the service. According to this table, two context services (namely the GPS and the network location service) offer the requested context scope (here: location) for the first scenario while characterizing the correct entity. Similarly there are seven context services, which potentially offer the requested information in scenario 2.

Context Query	Potential Context Offers
Scenario 1: Location (Address)	GPS (WGS84)
	Network location (WGS84)
Scenario 2: Activity (ActivityKoseRep)	Activity recognition 1 (ActivityKwapiscRep)
	Activity recognition 2 (ActivityLeeRep)
	Activity recognition 3 (ActivityWangRep)
	Activity recognition 4 (ActivityKoseRep)
	Activity recognition 5 (ActivityYanRep)
	Calendar Activity (ActivityCalendarRep)
	Calendar Events (EventDescription)

**Table 11.8:** Demonstrator A: Potential Context Offers for Scenario 1 and 2.

It is visible that in the first scenario none of the potential context services provide the information in the requested representation. In the second scenario it is only the context service *Activity recognition 4* that offers the correct representation. The data conversion from the representation offered by the potential services to the requested representation is tested by the next phase of the context offer and request matching, the mediation check. Without any *Inter-Representation Operations* and *metadata operators*, the mediation check would not be able to provide at least one chain for listener 1 and it would result in exactly one chain for listener 2, namely a chain using the service ‘Activity recognition 4’ as input provider. Table 11.9 shows the implemented IROs and Table 11.10 the implemented operator.

IRO Name	Scope	InputRep	OutputRep	Input Metadata	Output Metadata
Activity-KwapiscTo-LeeIRO	Activity	Activity-KwapiscRep: String (walking, jogging, ascendingStairs, descendingStairs, sitting, standing)	ActivityLeeRep: String (standing, walking, running, stairUpDown)		

IRO Name	Scope	InputRep	OutputRep	Input Metadata	Output Metadata
ActivityLee-ToKoseIRO	Activity	ActivityLee-Rep: String (standing, walking, running, stairUpDown)	ActivityKoseRep: String (walking, sitting, running, standing)		
SecondTo-MilliSecond-IRO	Scope	DoubleRep & Unit=second	DoubleRep & Unit=milli-second		
WGS84To-AddressIRO	Location	Location-WGS84 (Latitude, Longitude, Altitude)	AddressRep (Street, Number, City, Country)	Estimated-SensorCost (DoubleRep)	Estimated-SensorCost (DoubleRep)
CastIRO	Scope	Basic-Representation	Basic-Representation		

**Table 11.9:** Demonstrator A: Inter-Representation Operations

In Table 11.9, the CastIRO has to be highlighted. In difference to the other IROs, which are specific IROs, the CastIRO is a generic IRO (see Section 6.4.1). Specific IROs are exactly described in the ontology with respect to the scope and the input and output representations, while the description of a generic IRO only references superclasses for scope, input, and output representation. The respective generic IRO has to be called explicitly to check whether a conversion is applicable or not. In our case, the CastIRO is a generic IRO encapsulating Java type casts for the different atomic representations like String, Integer, or Boolean.

Metadata Operation	Input	Output
FreshnessInSecondsBasedOn-UnixTimeOperation	TimeStamp (Time_UnixTimeStamp)	Freshness (Double & Unit=second)

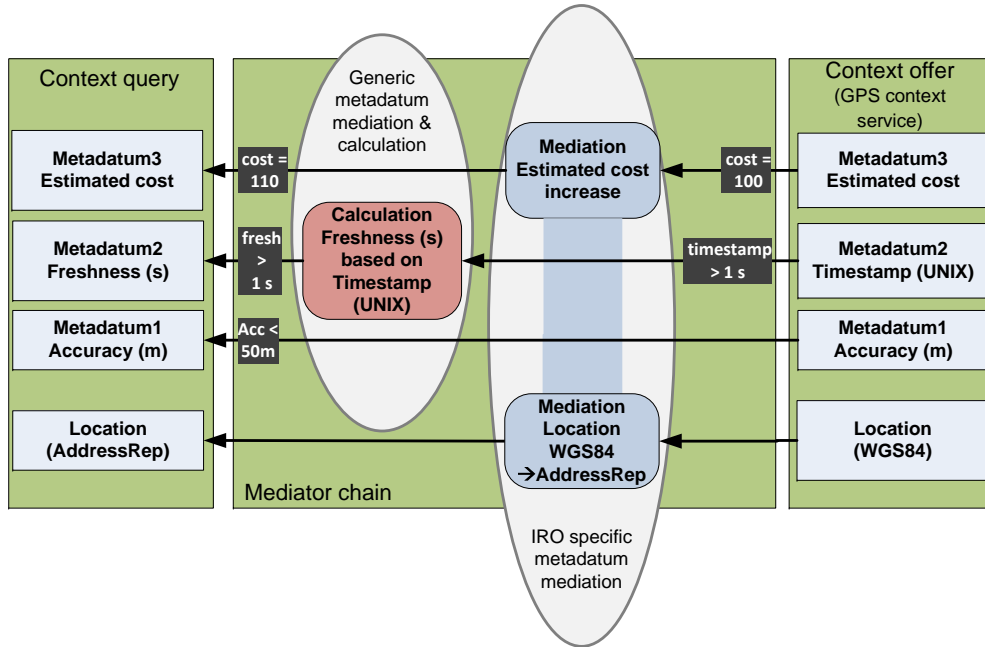
**Table 11.10:** Demonstrator A: Metadata Operation

With these IROs and the Metadata Operator, the mediation check results in a set of mediation chains, which are able to convert between a context service and a context listener. Figure 11.1 shows the resulting mediator chains for the first scenario. These chains look similar for both potential context providers: the information provided by the context service and represented as a WGS84 coordinate is transformed by the *WGS84ToAddressIRO* to the requested representation, namely an address. This conversion increases the cost, in this case the estimated cost<sup>13</sup> is increased by 10. After this conversion, the actual data are represented correctly. However, the constraint exists that the freshness has to be less than 10 seconds and neither the original offer nor the offer

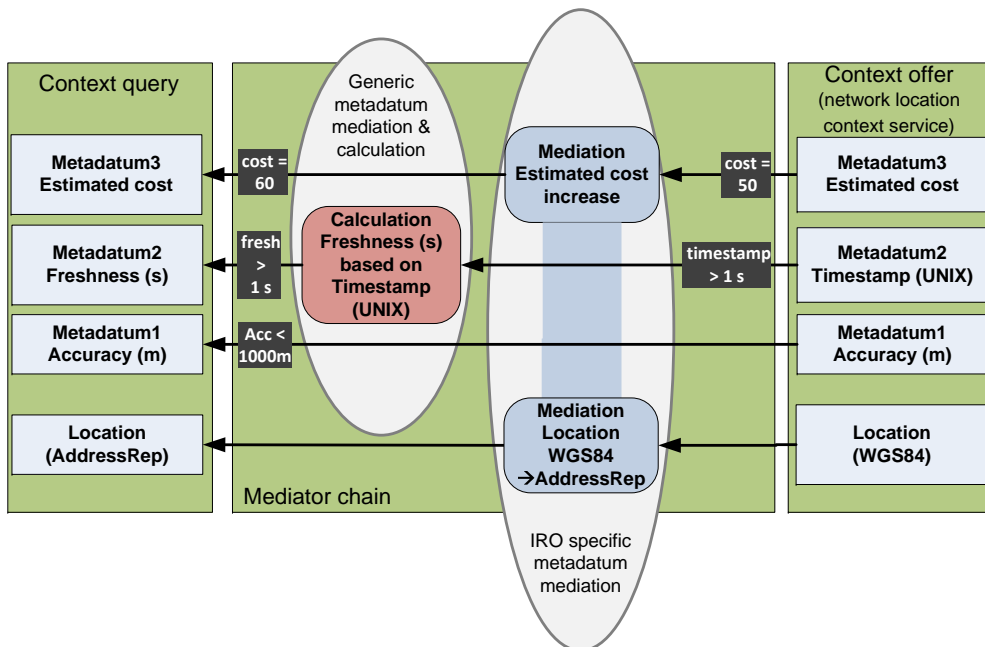
<sup>13</sup>In this demonstrator we only work with a unit-less estimation of the cost in order to demonstrate the concepts. It is part of the future work to provide concrete measurement methods to precisely measure the resource usage and provide precise cost estimation.



after the conversion by the IRO provides metadata of the type *freshness*. As this type can be provided by the metadata operator *FreshnessInSecondsBasedOnUnixTimeOperation*, this operator is appended to the chain. Now all data are available and can be converted into the requested representations.



(a) Chain 1: GPS Context Service → Context Query 1.



(b) Chain 2: Network Location Context Service → Context Query 1.

**Figure 11.1:** Demonstrator A: Chains in Scenario 1.

The screenshots depicted in Figure 11.2 show the visualization application of scenario 1 of Demonstrator A. In the subfigure (a), it is visible that the context query by the first listener (it is the first context query in the screenshot) is resolved (indicated by the thumb up) by two mediator chains (expressed by the number below this thumb). The subfigure (b) shows one of these two chains. The indicated chain is also shown in subfigure (b) of Figure 11.1. The thumb here indicates that this chain has been selected. An explanation will follow in the next section.

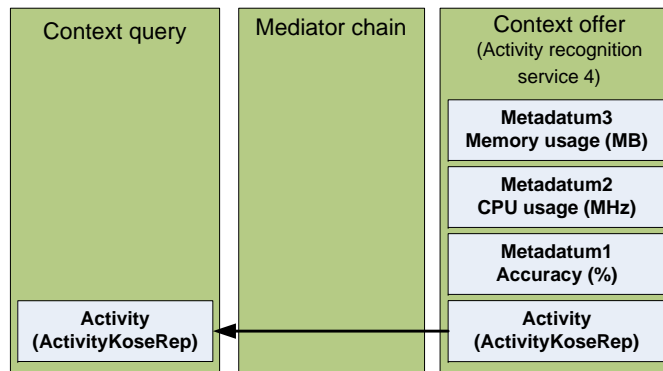


(a) Query 1: Location (AddressRep)

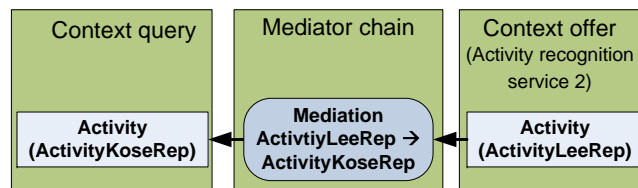
(b) Selected Chain: Network Location Service → Query 1

**Figure 11.2:** Demonstrator A: Screenshots of the Visualization Application in the first Scenario.

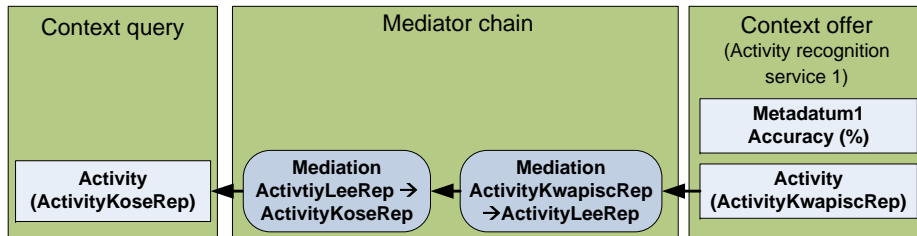
In difference to the first scenario, the context listener in the second scenario does not express any constraints and selection criteria. As a result the mediation process can completely focus on transforming the actual data into the correct representation. The results are depicted in Figure 11.3. As said before, the context service *Activity recognition 4* provides already the requested representation. As a consequence, the mediator chain is only contains an identity mediator as explained in Section 8.2. Additionally to this simple chain, it is possible to establish two more chains. Chain 2 consists of only one IRO mediator, whereas Chain 3 contains two mediators in series. Chain 1 and 3 are also depicted in the screenshots in Figure 11.4.



(a) Chain 1: Activity Recognition 4 → Context Query 2.



(b) Chain 2: Activity Recognition 2 → Context Query 2.



(c) Chain 3: Activity Recognition 1 → Context Query 2.

**Figure 11.3:** Demonstrator A: Chains in the Second Scenario

With this first Demonstrator A the ability to handle heterogeneously represented information including the conversion and the generation of missing metadata has been presented. Unlike existing approaches the established mediator chains have been generated fully autonomously by the middleware. Implicitly this demonstrator also provides an example that our approach is also able to handle loosely coupled context services and consumers as requested in Requirement 2 described by offers and queries (as described in Requirement 4).

## 11.4 Demonstrator B: Simple Selection

After the successful matching of at least one context offer regarding a context query and hence the establishment of at least one mediator chain, the middleware has to select the best chain with respect to the expressed selection criteria in the context query. In general, the selection approach does not only try to find a solution query by query. It rather tries to find a set of mediator chains, which satisfy all context requests.



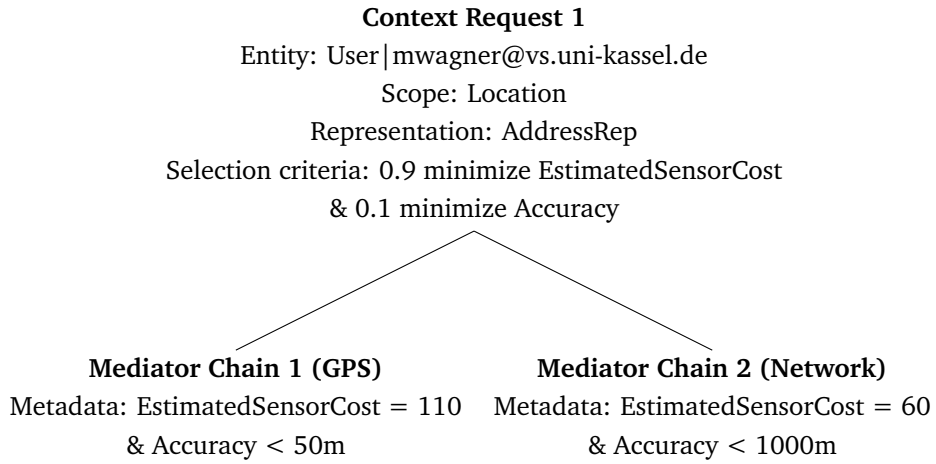
(a) Query 2: Activity (ActivityKoseRep). (b) Selected Chain: Activity Recognition 1 → Context Query 2.

**Figure 11.4:** Demonstrator A: Screenshot of the Visualization Application for the Second Scenario.

For this reason, we register in every scenario of this Demonstrator B only one request<sup>14</sup>. The multi-selection approach selecting a set of context services for a set of context queries is demonstrated afterwards in the next section. In this section, the selection for a set of context queries consisting of only one query is demonstrated in two different scenarios. Each scenario presents the selection of a mediator chain for one context query. In the scenarios of this demonstrator we reuse the two queries already used in the scenarios of the first *Demonstrator A*.

In the first scenario of this Demonstrator B, the context query requests the location of the user *mwagner@vs.uni-kassel.de*. This location should be represented as an address. As depicted in Table 11.7, the potential context provider has to offer the requested information with a freshness of 10 seconds. The context consumer expressed a high importance on minimizing cost (0.9) while weighting the importance of the minimization of accuracy to 0.1. After the matching phase (see first scenario of Demonstrator A), two mediator chains could be established as shown in Figure 11.5.

<sup>14</sup>More precisely, we started only one context listener which registers its query to the system. There are additionally several context queries registered from the different context services implementing reasoning mechanisms (see Table 11.6).



**Figure 11.5:** Demonstrator B: Potential Mediator Chains in the First Scenario

For both options the aggregated utility function (AUF, see Section 9.2.2) has to be calculated. As in this case a potential selection set only consists of one chain, the AUF is a linear combination of the single utility function (SUF, see Section 9.2.2) and the cost minimization utility function (CMUF, see Section 9.2.2):

$$u(X) = (1 - \alpha) \cdot u_r(X) + \alpha \cdot \frac{\sum_{d \in C} u_c(X, d)}{|C|} \quad (11.1)$$

As described in Section 9.2.2, the parameter  $\alpha$  is used to have at least minimal influence on the utility with regard to cost minimization even if in the selection criteria doesn't express any preference with regard to any cost dimension. In our system  $\alpha$  has been set to 0.05.

As depicted in Table 11.7, Context Listener 1 specified two selection criteria in his request. Hence, the SUF is a weighted sum of the two metadata dimensions *EstimatedSensorCost* and *Accuracy*. The values for both dimensions are normalized. For the normalization we have to distinguish between quality related metadata and cost related metadata. Quality related metadata are normalized by the maximum value used in a context offer for the respective quality dimensions. In contrast to the normalization of the cost related metadata dimensions, first the cost caused by the set of all available mediator chains are calculated and these values are used afterwards for the normalization. For all normalizations, the denominator is increased with a very small  $\epsilon$  (here  $\epsilon = 0.00000001$ ). The SUF for the first mediator chain  $M1$  looks as follows:

$$\begin{aligned} u_r(M1) &= 0.1 \cdot \left(1 - \frac{50}{1000 + \epsilon}\right) + 0.9 \cdot \left(1 - \frac{110}{60 + 110 + \epsilon}\right) \\ &= 0.095 + 0.318 \\ &= 0.413 \end{aligned} \quad (11.2)$$

For the CMUF, the utility values for the different cost dimensions are aggregated and normalized by the number of cost dimensions, which are specified in the ontology. In

our case we have only specified four cost dimensions in the ontology<sup>15</sup>. Hence the CMUF for mediator chain 1 looks as follows:

$$\begin{aligned} cmuf(M1) &= \frac{1 - \frac{110}{60 + 110 + \epsilon}}{4} \\ &= 0.088 \end{aligned} \quad (11.3)$$

This results then in the following AUF:

$$\begin{aligned} u(M1) &= (1 - \alpha) \cdot u_r(M1) + \alpha \cdot cmuf(M1) \\ &= 0.396 \end{aligned} \quad (11.4)$$

Similarly, the SUF, CMUF, and AUF for the second mediator  $M2$  chain are calculated:

$$\begin{aligned} u_r(M2) &= 0.1 \cdot \left(1 - \frac{1000}{1000 + \epsilon}\right) + 0.9 \cdot \left(1 - \frac{60}{60 + 110 + \epsilon}\right) \\ &= 1 \cdot 10^{-12} + 0.582 \\ &= 0.582 \end{aligned} \quad (11.5)$$

$$\begin{aligned} cmuf(M2) &= \frac{1 - \frac{60}{60 + 110 + \epsilon}}{4} \\ &= 0.162 \end{aligned} \quad (11.6)$$

$$\begin{aligned} u(M2) &= (1 - \alpha) \cdot u_r(M2) + \alpha \cdot cmuf(M2) \\ &= 0.561 \end{aligned} \quad (11.7)$$

The chain set<sup>16</sup> resulting in the highest utility is then selected and the respective context service providing the offer, which serves as input provider for at least one of the selected chains, are activated. The logging output for the previously described demonstrator is depicted in Listing 11.1.

```

1 Start selection process
2 #Queries=1
3 #QueriesForReasoner=0
4 #Chains=2
5 #AverageChainsPerQueries=2
6 #Combinations = 2
7 Checking combination #1 of 2
8 calculateSingleUtilityFunction for query=query_position_demo1
  -1343829445822-75
9 Calculated utility for query[query_position_demo1
  -1343829445822-75] and chainSet=[m_offer_ANDROID_GPS_42
  -1343829444616-4_query_position_demo1
  -1343829445822-75-1343829452632-96;] = maxUtility
  (0.095000000000005) + costUt(0.3176470588235294) =
  0.4126470588235794

```

<sup>15</sup>These cost dimensions are all used in the different context offers (see Section 11.2) and are defined in the context ontology: CPUUsage, DeviceMemoryUsage, EstimatedSensorCost, and PowerConsumption.

<sup>16</sup>In this scenario, the set consists of only one chain.

```

10 calculateAggregatedUtilityFunction for set of chains (
    m_offer_ANDROID_GPS_42-1343829444616-4_query_position_demo1
    -1343829445822-75-1343829452632-96;) = 0.39642647096328276
11 Checking combination #2 of 2
12 calculateSingleUtilityFunction for query =query_position_demo1
    -1343829445822-75
13 Calculated utility for query[query_position_demo1
    -1343829445822-75] and chainSet=[
    m_offer_ANDROID_NetworkLocation_42-1343829444527-91
    _query_position_demo1-1343829445822-75-1343829452937-82;] =
    maxUtility(1.000000082740371E-12) + costUt
    (0.5823529411764705) = 0.5823529411774705
14 calculateAggregatedUtilityFunction for set of chains (
    m_offer_ANDROID_NetworkLocation_42-1343829444527-91
    _query_position_demo1-1343829445822-75-1343829452937-82;) =
    0.5613235297877146
15
16 Selection finished in 8ms: Selected chains = {Chain(Offer=
    offer_ANDROID_NetworkLocation_42-1343829444527-91;Query=
    query_position_demo1-1343829445822-75):Location|->
    LocationWGS84->AddressRep,} utilityOfSelectionSet =
    0.5613235297877146
17

```

**Listing 11.1:** Demonstrator B: Logging Output of the Selection Service in the First Scenario

The results in the logging data of AUF, SUF, and CMUF are the same as in the previously described theoretical calculations. The chain set with the highest AUF is selected. In our case, the process ends with the selection of the second chain and hence in the activation of the offer *Network location*. Which context services, context offers and mediator chains are selected respectively activated is also shown in the visualization tool. The screenshot in Figure 11.6 presents the demonstrator after finalizing the selection (The left subfigure shows the selected chain and the right subfigure the activated offer). The thumb up indicates that the selection respectively activation happened as calculated before.

In difference to the first scenario, the matching process for the context query in the second scenario results in three mediator chains (see Demonstrator A). Additionally this query does not express any selection criteria. But as the context providers of the different chains are context services implementing a context reasoner, the respective providers are requiring again some context information to provide the requested information. In our example, all context providers are activity reasoners retrieving the activity from the acceleration of the device. For this purpose they request for acceleration, which is provided by two context offers. The results of the matching process for all requests are depicted in Figure 11.7.

There are three mediator chains available which can potentially provide the requested information for the context query in this second scenario of this demonstrator. Each of these chains is again connected to a context query. This query is expressed by the respective context service serving as input provider for the respective chain. For each of these context queries, again two mediator chains are provided.

The establishment of the different mediator chains is the result of the matching process. This matching process is called for all context queries registered to the system. The



(a) Selected Chain: Network Location Service → Query 1

(b) Offer: Network Location

**Figure 11.6:** Demonstrator B: Screenshots of the Visualization Application in the First Scenario

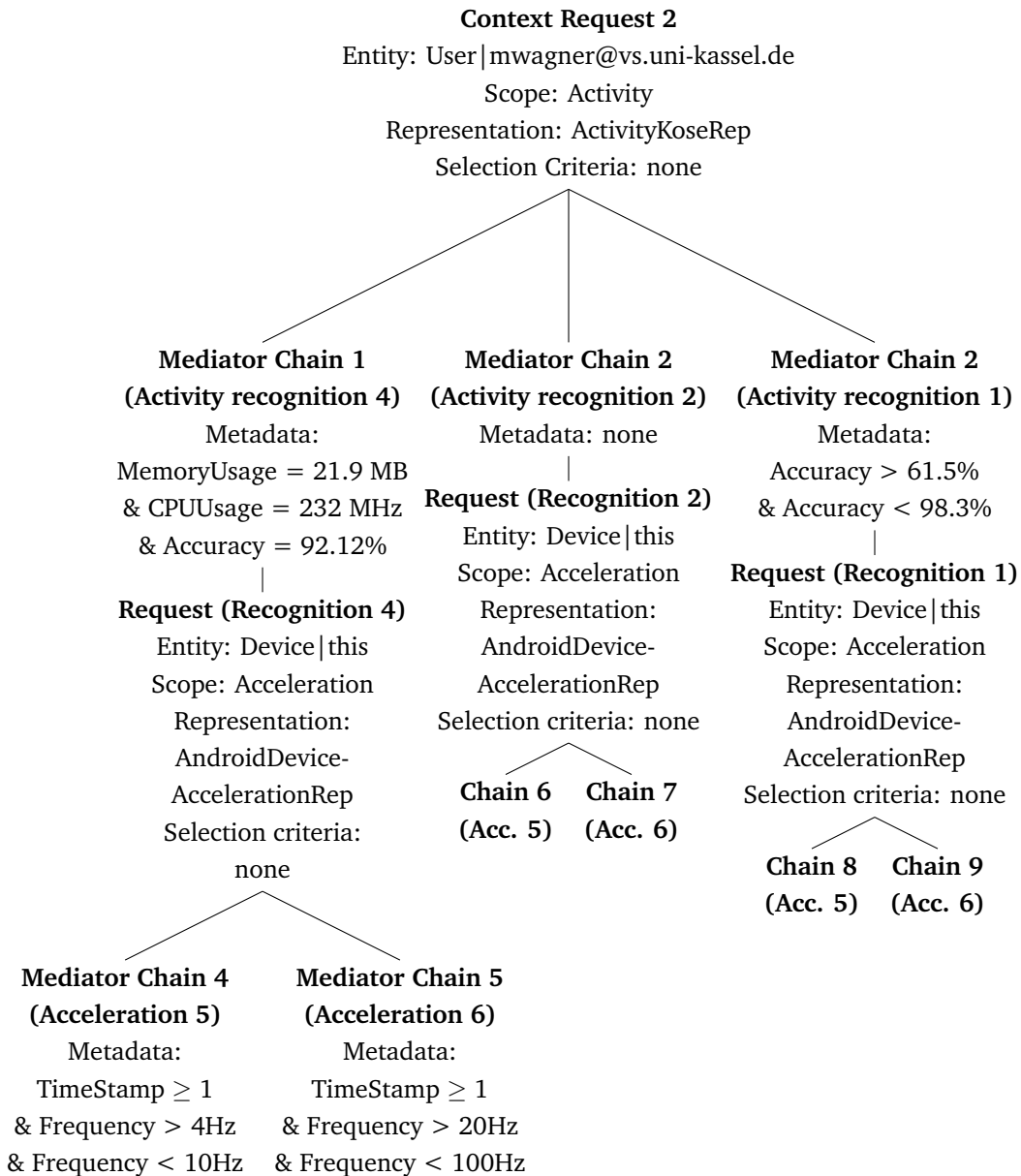
actual matching process does not differ for requests by context listener (or context-aware application in general) or requests by context services. Mediator chains resolving a request of a regular context listener are immediately forwarded to the selection. In contrast, mediator chains for requests of context services are only considered during the selection of the corresponding services serve as input provider for at least one chain.

The selection algorithm described in Section 9.4 is now used to build combinations of mediator chains, for which the AUF is calculated and which build potentially the set of chains serving as input providers. The algorithm starts with adding one mediator chain for every query (which is regular and hence not expressed by a context reasoner) to a set. For this set, it is checked in the second phase if a context reasoner is used as input provider for one or more chains. If this is the case, for every query of the reasoners, again a chain is added to the set. This is repeated for all combinations of chains respectively reasoner queries. For every of these combinations the AUF is calculated. In Figure 11.7 it is easy to see that every path within the depicted tree builds a valid combination. Hence, the AUF has to be calculated for the six chain sets  $\{Chain1, Chain4\}$ ,  $\{Chain1, Chain5\}$ ,  $\{Chain2, Chain6\}$ ,  $\{Chain2, Chain7\}$ ,  $\{Chain3, Chain8\}$ , and  $\{Chain3, Chain9\}$ .

As neither the query of the context listener in this second scenario nor the queries of the different context services specify any selection criteria, the SUF always results in 0.5. Plus the CMUF results in the same value as only the context service *Activity recognition 4* defines some cost related metadata constraints. Thus the AUF is 0.95000005 for every



of the six valid combinations of mediator chains. As a result, the selection is finally a random selection from the set of valid combinations<sup>17</sup>. The complete logging output for the second query can be found in Listing A.1 in the appendix.



**Figure 11.7:** Demonstrator B: Potential Mediator Chains/Offers in the Second Scenario

With this second demonstrator, the ability of our system to select a mediator chain according to the needs of a context listener has been shown as requested in Requirement 6. Additionally it has been demonstrated that the specification of selection criteria is optional and that the system is also able to proceed without these criteria. In general, the specification of metadata constraints, with or without those criteria, is enough to filter out non-fitting services respectively offers.

<sup>17</sup>The first valid combination for which the AUF is calculated, is finally chosen.

## 11.5 Demonstrator C: Multiple Selection

With the Demonstrator B, the selection of a context service for only one context requests has been demonstrated. To demonstrate the usefulness of our selection concepts for a whole set of context queries, we introduce several more context queries, which are depicted in Table 11.11. These queries and also the two queries used in Demonstrator A and B are registered simultaneously at the middleware. Along with to these six queries, the requests of the context services, as depicted in Table 11.6, are registered.

Context Consumer	Entity	Scope	Representation	Metadata constraints	Selection criteria
Context listener 3	User   mwagner@vs.uni-kassel.de	FallDetection	BooleanRep		
Context listener 4	Device   this	BatteryLoad	PercentRep		
Context listener 5	Device   this	Acceleration	AndroidDevice-AccelerationRep		
Context listener 6	User   mwagner@vs.uni-kassel.de	Activity	EventDescription-Rep		

**Table 11.11:** Demonstrator C: Context Queries

The matching process results in 25 mediator chains. These chains are shown in Table 11.12. As already explained in the previous Demonstrators A and B, there are two potential offers for context listener 1 and three offers for listener 2. The rest of this table can be interpreted analogously.

Context Query	Context Offer
<b>Context listener 1: Location</b>	<b>Network location</b>
	GPS
<b>Context listener 2: Activity (Kose Rep)</b>	<b>Activity recognition 4</b>
	Activity recognition 1
	Activity recognition 2
<b>Context listener 3: Fall detection</b>	<b>Fall detection</b>
<b>Context listener 4: Battery load</b>	<b>Battery: Battery load</b>
<b>Context listener 5: Acceleration</b>	<b>Accelerometer (Other devices SENSOR_DELAY_NORMAL)</b>
	Accelerometer (Other devices SENSOR_DELAY_FASTEST)
<b>Context listener 6: Activity (Event description)</b>	<b>Calendar: Activity (Event description)</b>
<b>Fall detection: Acceleration</b>	<b>Accelerometer (Other devices SENSOR_DELAY_NORMAL)</b>
	Accelerometer (Other devices SENSOR_DELAY_FASTEST)

Context Query	Context Offer
Activity recognition 1: Acceleration	Accelerometer (Other devices SENSOR_DELAY_NORMAL)
	Accelerometer (Other devices SENSOR_DELAY_FASTEST)
Activity recognition 2: Acceleration	Accelerometer (Other devices SENSOR_DELAY_NORMAL)
	Accelerometer (Other devices SENSOR_DELAY_FASTEST)
Activity recognition 3: Acceleration	Accelerometer (Other devices SENSOR_DELAY_NORMAL)
	Accelerometer (Other devices SENSOR_DELAY_FASTEST)
<b>Activity recognition 4: Acceleration</b>	<b>Accelerometer (Other devices SENSOR_DELAY_NORMAL)</b>
	Accelerometer (Other devices SENSOR_DELAY_FASTEST)
Activity recognition 5: Acceleration	Accelerometer (Other devices SENSOR_DELAY_NORMAL)
	Accelerometer (Other devices SENSOR_DELAY_FASTEST)
Activity recognition 3: Location	Network location
	GPS
Activity recognition 3: Microphone log	Microphone

**Table 11.12:** Demonstrator C: Mediator Chains

As depicted, there are several alternative mediator chains for most context queries. In total there are 48 different valid combinations of mediator chains: for the first query, there are 2 options. For the second query there are three chains but all are using a context reasoner having each 2 different variants. Hence there are in total six different combinations for query 2. Query 3 has only one chain which is starting from a reasoner for which again, two alternative chains are available. Query 4 and 6 have only one chain each and query 5 has two alternatives. Thus there are in total  $2 \cdot 6 \cdot 2 \cdot 1 \cdot 2 \cdot 1 = 48$  valid combinations of mediator chains. The result of the selection process is also depicted in the previous table: the selected chains are highlighted bold.

With this demonstration we have again shown that our system holds the requirement for a selection based on CoC and QoC (see Requirement 6). Furthermore it demonstrates the ability to share context services. In this demonstrator, the context service ‘Accelerometer’ with its offer ‘Other devices SENSOR\_DELAY\_NORMAL’ is used simultaneously by context listener 5 and serves as input provider for the context services ‘Fall detection’ and ‘Activity recognition 4’. By sharing services, the system actively reduces the resource consumption as requested in Requirement 7.

## 11.6 Demonstrator D: Discovery of Remote Offers

In the previous sections, the discovery, matching, and selection of local context services have been discussed intensively. However, one additional benefit of our approach is that we can share context information not only locally but also with other devices. Context offers are advertised immediately after the respective local context services have been selected and activated so that they can be discovered by remote devices. In general we only advertise these context services, which are also used locally. The reason for that is, that we mainly focus on mobile devices with limited energy resources. It follows that only if we have a benefit by sharing information (namely we can also use the provided information<sup>18</sup>), we allow to share a context service<sup>19</sup>.

For this demonstrator we installed and started the middleware and context consumers as described in the previous Demonstrator C on a first device. Additionally, we started a second instance on a second device whereby all context services from this instance have been removed and this device has no local context service available. After at maximum of 10 seconds<sup>20</sup>, the context offers advertised by the first device are discovered by the second device and the matching started. The first device offers all offers that are selected as described in Demonstrator C. Thereby it has to be highlighted that several of these offers characterize the first device (the characterized entity is *Device | this* whereby *this* is replaced by the ID of the executing device). Hence for these offers it is not possible to establish any mediator chains on the second device. In contrast, information characterizing the user *mwa@vs.uni-kassel.de* can also be reused on the second device and according mediator chains could be established.

Figure 11.8 shows screenshots of the visualization application running on the second device. In part (a) some of the discovered context offers are shown. The thumb up indicates that the respective offer serves as input provider of at least one chain. The 'e' under the thumb marks external offers<sup>21</sup>. The number under the 'e' is the number of the bound mediator chains. Part (b) shows some of the context queries registered on the second device. The thumb down and the zero under this thumb indicate that there is no mediator chain for the first shown query regarding the battery load, whereas there is one chain per query for query 2 and 3 (indicated by the thumb up and the 1 under this square). As it can be seen in the two screenshots, we use the IP address of the device as its identifier. Hence we can see that the remote offers describing a device (e.g. the second offer in Figure 11.8 (a)) refer to another entity as in the request of the second device (e.g. in the first query in Figure 11.8 (b)).

With this demonstration we gave an example for the remote discovery of context offers as requested in Requirement 1. Furthermore, this would not be possible without fulfilling the requirements on loosely coupled service (see Requirement 2) and on the semantic description of these service as well (see Requirement 4).

---

<sup>18</sup>The local usage is currently the sole criteria for the remote advertisement of a context service. Other criteria like a user-defined privacy policy are part of the future work.

<sup>19</sup>Nevertheless, the middleware has also a flag which enables the sharing of all context services. This is mainly relevant for devices without energy limits, like servers or for mobile devices which are currently charging.

<sup>20</sup>This is the current remote discovery cycle.

<sup>21</sup>This is only for visualization purposes. Apart from the binding process, the system does not distinguish between local and external context offers.



(a) Offers Discovered on the Second Device (b) Queries Registered at the Second Device.

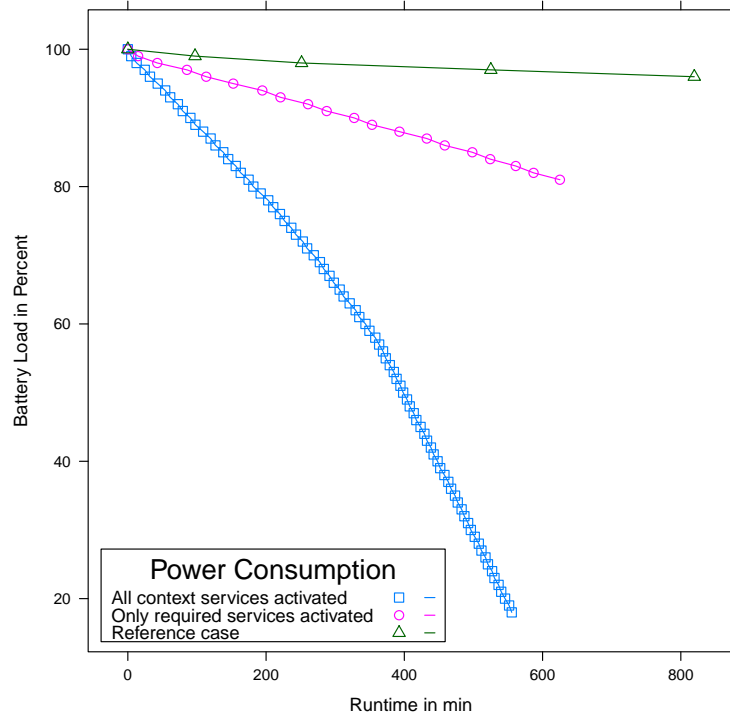
**Figure 11.8:** Demonstrator D: Remote Discovery of Context Offers on the Second Device

## 11.7 Demonstrator E: Cost Minimization

Cost minimization is an important issues of this work. For this purpose context services are deactivated if they are not required respectively services are activated if one or more mediator chains are selected by the selection algorithm using a context service as input provider. To measure the benefits of our work, we have made evaluations with three different configurations. The first measurement setup is inspired by Paller [103], who made some measurements to retrieve the energy consumption of an acceleration sensor. This first measurement serves as a reference measurement and shows the energy consumption of the device with nearly no activity disregarding the application implemented for logging the battery load. The device is started in the airplane mode, which means that all communication functionalities (like 3G, WiFi, Bluetooth, . . .) are switched off. The device is fully charged and stayed undisturbed for at least 8 hours meanwhile the logging application records the battery load.

In the second configuration, the device is again fully charged and also connected to 3G and WiFi networks. Thereby our middleware and the full demonstrator, which means all context services as described in Section 11.2, are activated and running on the device. Additionally, all context queries used in the previous demonstrators are registered to the middleware. Additionally the visualization application is running. Again the device is running for at least 8 hours.

The third configuration is similar to the second configuration with the difference that all context services are activated immediately after the registration. This can be configured in the middleware by a system flag called *activateAfterDiscovery*. The results of this evaluation are visualized in Figure 11.9.



**Figure 11.9:** Demonstrator E: Power Consumption in Different Use Cases

From Figure 11.9, it is obvious that solely activating the required context services reduces the energy consumption enormously. While in the case, in which all services are immediately activated, the battery load is decreased to 18% after 555 min in average, the system, which only activates the required services decreases the battery to 83–84% of the battery load in the same time. For sure, the effect strongly depends on the context queries and the according matching results. For example, if the first listener would prioritize the minimization of the accuracy<sup>22</sup> instead of minimizing cost, the GPS sensor would be the first choice, and effect of cost minimization would not be that good as in this demonstrator. However as long as not all context services would be necessary to fulfil the context requests, deactivation of irrelevant service would result in minimization of the energy consumption. With this demonstration, we clearly showed the benefits of activation/deactivation of context services as requested in Requirement 5. Moreover, this demonstration is another good example how our system fulfils the requirement on reducing the resource consumption (see Requirement 7).

<sup>22</sup>Depending on the representation of the accuracy which is a radius in meter: the smaller the accuracy, the better the result is.

## 12 Performance and Scalability

---

*The birth of an idea is that happy moment when everything appears possible and reality has not yet entered into the problem.*

– Rudolf Diesel (1858-1913)

Written shortly after one of his early engine models had blown up and nearly killed him.

Besides the demonstration of the practical usability of our concepts, the performance and scalability of them is an important aspect. While the theoretic complexity of the most important algorithms has been discussed in the respective chapters, we will now focus on practical experiments on a mobile device and reflect the results on the theoretical discussions. For all evaluations we have used a Motorola Xoom tablet<sup>1</sup>. This device is equipped with a 1 GHz dual-core processor and 1 GB RAM. The installed operating system is Android 4.0<sup>2</sup>.

### 12.1 Mediation Service

This first analysis focuses on the performance and scalability of the mediation service. The mediation service is executed during the matching process (see Chapter 8) and is responsible for the mediation check as described in Section 8.2. The mediation check is called after the initial matching, which means that a matching offer and query at least refer to the same entity (type) and the same context scope. The result of this check is a (potentially empty) set of mediator chains mediating the context information provided by the context service into the requested representation. A mediator chain consists of a sequence of Inter-Representation Operations (IROs) and metadata operations.

Interpreting representations as vertices and IROs respectively metadata operations as edges, the mediation problem can be transferred to a path finding problem within a directed graph. Mediator chains are simple paths in this graph starting from the representation offered by the context service to the representation requested by the context consumer. The performance of the mediation check mainly depends on the number of available IROs, metadata operations, and on the structure of the resulting graph.

In this evaluation we focus on the performance and scalability of the mediation service depending on the number of available IROs. In order to prove the scalability in the worst

---

<sup>1</sup>[http://www.motorola.com/us/consumers/MOTOROLA-XOOM/72804\\_en\\_US\\_pd.html](http://www.motorola.com/us/consumers/MOTOROLA-XOOM/72804_en_US_pd.html), last visited on Aug 08, 2012.

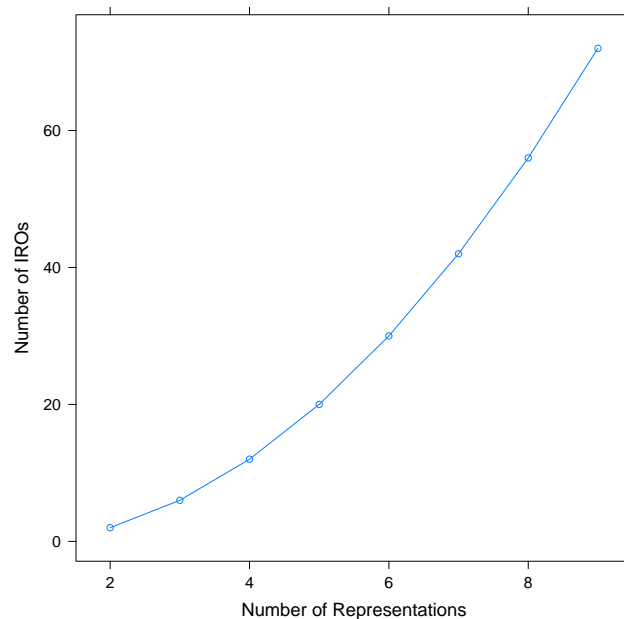
<sup>2</sup>Android Ice Cream Sandwich <http://www.android.com/about/ice-cream-sandwich/>, last visited on Aug 08, 2012.

<b>Input \ Output</b>	Rep 1	Rep 2	Rep 3	Rep 4
Rep 1	–	IRO 1	IRO 2	IRO 3
Rep 2	IRO 4	–	IRO 5	IRO 6
Rep 3	IRO 7	IRO 8	–	IRO 9
Rep 4	IRO 10	IRO 11	IRO 12	–

**Table 12.1:** Scalability Evaluation: Generated IROs for Four Representations

case, each representation should be reachable by an IRO from each other representation. For this purpose, we developed a factory, which generates a set of IROs based on a set of representations. This factory generates all possible IROs between the different representations except IROs having the same representation as input and output, hence providing the identity. The factory generates for  $n$  representation  $n^2 - n$  IROs. Table 12.1 shows this for four representations.

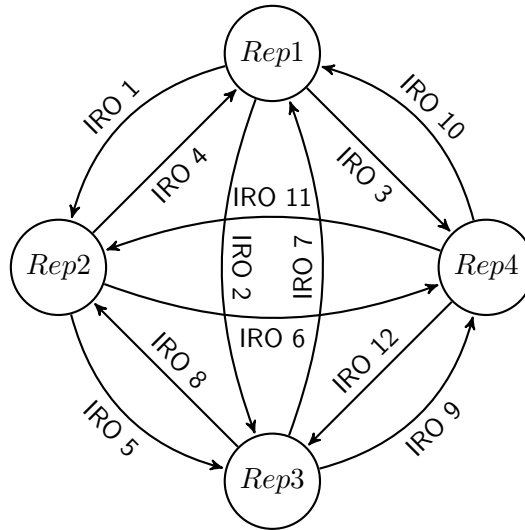
Figure 12.1 visualizes the dependency of the number of generated IROs compared to the number of representations.



**Figure 12.1:** Scalability Evaluation: Number of Representations Compared to Generated IROs

The generated IROs are registered at an instance of the mediation service. Then the *mediation check* is called for a context query (Entity: *Ent*, Scope: *Scope*, Representation: *Rep n*) and a context offer (Entity: *Ent*, Scope: *Scope*, Representation: *Rep 1*). The time is measured for establishing all possible mediator chains mediating between the offered *Rep 1* and the requested *Rep n*. For this evaluation we did not limit the number of maximal results and also not the number of hierarchies as described in Section 8.5.





**Figure 12.2:** Scalability Evaluation: Resulting Graph for Four Representations

Figure 12.2 shows the resulting graph for four representations and the generated IROs. Interpreting representations as vertices and IROs as edges of a directed acyclic graph allows to redefine mediator chains as paths from a certain node  $s$  to a certain node  $t$ . As stated by Sloane,  $a(n) = \sum_{k=0}^n \frac{n!}{k!}$  “is [...] the number of paths (without loops) in the complete graph on  $n+2$  vertices starting at one vertex  $v1$  and ending at another  $v2$ ” [126]. In the depicted example there are five simple paths from representation ‘Rep 1’ to ‘Rep 4’ (more precisely, there are five simple paths to every other representation):

- Rep 1  $\xrightarrow{IRO\ 1}$  Rep 2  $\xrightarrow{IRO\ 5}$  Rep 3  $\xrightarrow{IRO\ 9}$  Rep 4
- Rep 1  $\xrightarrow{IRO\ 1}$  Rep 2  $\xrightarrow{IRO\ 6}$  Rep 4
- Rep 1  $\xrightarrow{IRO\ 2}$  Rep 3  $\xrightarrow{IRO\ 8}$  Rep 2  $\xrightarrow{IRO\ 6}$  Rep 4
- Rep 1  $\xrightarrow{IRO\ 2}$  Rep 3  $\xrightarrow{IRO\ 9}$  Rep 4
- Rep 1  $\xrightarrow{IRO\ 3}$  Rep 4

To find these paths respectively the mediator chains, a deep-first search algorithm is used. Starting from the offered representation, it is called recursively for every node respectively representation connected by an IRO with the starting node. It terminates if

- (a) the requested representation is reached,
- (b) or the representation has already been visited by the paths which means that there is a cycle in the path (resulting in a cycle exception),
- (c) or if no IROs respectively edges start in the current vertex.

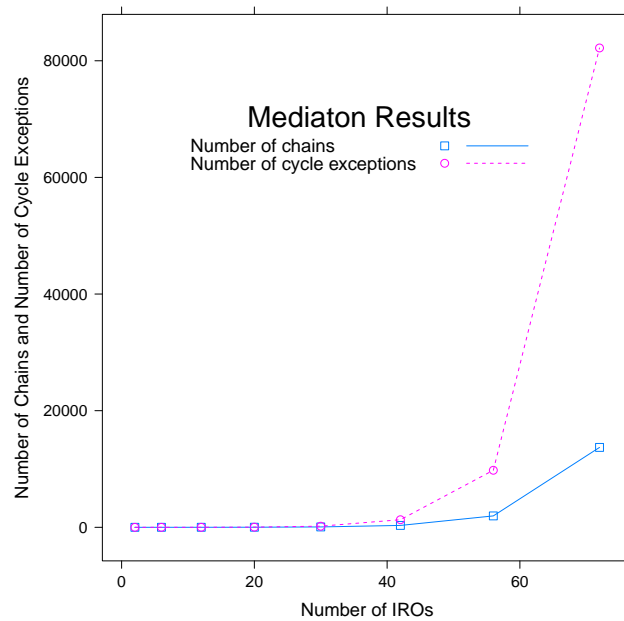
For our example with four representations, the algorithm results in the following six cycle exceptions additionally to the five chains:

- Rep 1  $\xrightarrow{IRO\ 1}$  Rep 2  $\xrightarrow{IRO\ 4}$  Rep 1
- Rep 1  $\xrightarrow{IRO\ 1}$  Rep 2  $\xrightarrow{IRO\ 5}$  Rep 3  $\xrightarrow{IRO\ 7}$  Rep 1

- Rep 1  $\xrightarrow{IRO\ 1}$  Rep 2  $\xrightarrow{IRO\ 5}$  Rep 3  $\xrightarrow{IRO\ 8}$  Rep 2
- Rep 1  $\xrightarrow{IRO\ 2}$  Rep 3  $\xrightarrow{IRO\ 7}$  Rep 1
- Rep 1  $\xrightarrow{IRO\ 2}$  Rep 3  $\xrightarrow{IRO\ 8}$  Rep 2  $\xrightarrow{IRO\ 4}$  Rep 1
- Rep 1  $\xrightarrow{IRO\ 2}$  Rep 3  $\xrightarrow{IRO\ 8}$  Rep 2  $\xrightarrow{IRO\ 5}$  Rep 3

In general, the algorithm generates in  $a(n) = \sum_{k=1}^n k \cdot k! \cdot \binom{n}{k}$  cycle exceptions with  $n = |Representations| - 2$ . According to Stephan, this is the “[...] the number of sequences – where each member is an element in a set consisting of  $n$  elements – such that the last member is a repetition of a former member” [132].

The description of the algorithm is simplified as we skipped the metadata mediation. This is described in detail in Section 8.5.



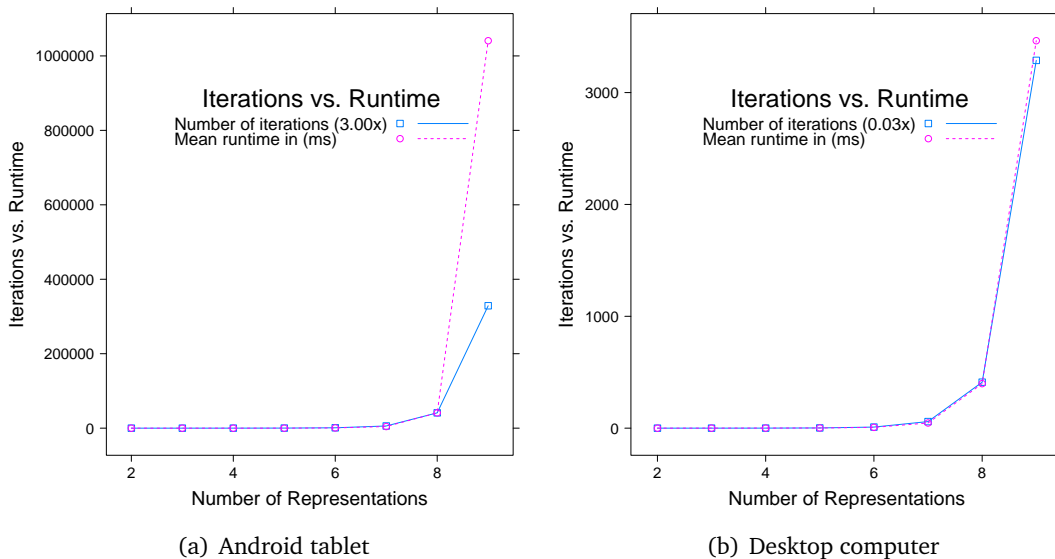
**Figure 12.3:** Scalability Evaluation: Number of Chains and Cycle Exceptions Depending on Number of IROs

Figure 12.3 shows the resulting number of mediator chains and cycle exceptions depending on the number of IROs. As described in Section 8.5, the algorithm is limited by the two system parameters `#maximalNumberOfHierarchies` and `#maximalNumberOfResults` and hence results in a complexity of  $\#maximalNumberOfHierarchies \cdot \#maximalNumberOfResults$ . However, as we are interested in the performance of the algorithm (independent of any limiting parameters) we set both parameters to infinite. The algorithm is then called  $a(n) = n \cdot (a(n - 1) + 1)$  times. “[...] for  $n \geq 1$ ,  $a(n)$  is the number of non-empty sequences with  $n$  or fewer terms, each a distinct element of  $1, \dots, n$ ” [127].

Representations	IROs	Chains	∅ Chain length	Cycle ex.	∅ Chain length (cycle ex.)	Iterations	n	Mean (ms)	Standard deviation (ms)
2	2	1	1.00	0	0.00	1	50	1.420	0.499
3	6	2	1.50	1	2.00	4	50	3.820	0.523
4	12	5	2.20	6	2.67	15	50	13.040	0.450
5	20	16	3.06	33	3.45	64	50	61.500	12.614
6	30	65	4.02	196	4.33	325	50	646.100	50.040
7	42	326	5.00	1305	5.25	1956	50	4374.760	69.913
8	56	1957	6.00	9786	6.20	13699	50	41082.820	145.054
9	72	13700	7.00	82201	7.17	109600	50	1040791.160	7589.211

**Table 12.2:** Runtime Statistics for Mediation

In this evaluation we measured the time for establishing all possible mediator chains mediating between the offered Rep 1 and the requested Rep n. The results of the evaluation are depicted in Table 12.2. These results are also visualized in Figure 12.4. Each evaluation has been repeated 50 times (as indicated by the n in Table 12.2). The third column indicates the number of established chains.



**Figure 12.4:** Scalability of the Establishment of Mediator Chains.

This evaluation also shows that the algorithm is calculating all possible mediator chains and correctly skipping chains containing cycles. In Figure 12.4 (a), we compare the mean runtime to the number of iterations (scaled by 3.00). From this diagram, we can see that the expected complexity and measured runtime correspond for all cases except the last test case with 9 representations. As the comparison of expected complexity

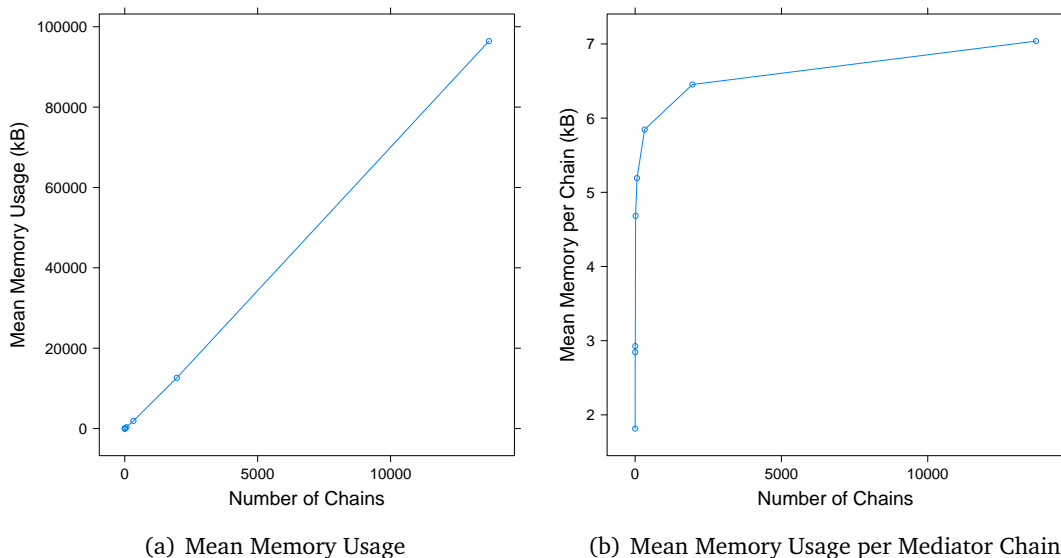
and measured runtime was not satisfying, we started the evaluation on another device, namely on a desktop computer equipped with an Intel i5 processor and 8 GB RAM. The resulting runtime curve is depicted in Figure 12.4 (b). The runtime is compared to the number of iterations (now scaled by 0.03). This analysis gives the expected results.

In order to analyse the discrepancy of expected runtime and measured runtime on the first device (the Android tablet Motorola Xoom), we also analysed the memory usage. The results of the memory analysis are depicted in Table 12.3 and Figure 12.5.

Representations	IROs	Chains	Ø Chain length	n	Mean (kB)	Standard deviation (kB)
2	2	1	1.00	50	1.816	0.756
3	6	2	1.50	50	5.691	1.070
4	12	5	2.20	50	14.638	42.391
5	20	16	3.06	50	74.932	31.742
6	30	65	4.02	50	337.509	6.539
7	42	326	5.00	50	1906.061	57.119
8	56	1957	6.00	50	12628.939	283.769
9	72	13700	7.00	50	96423.219	1661.985

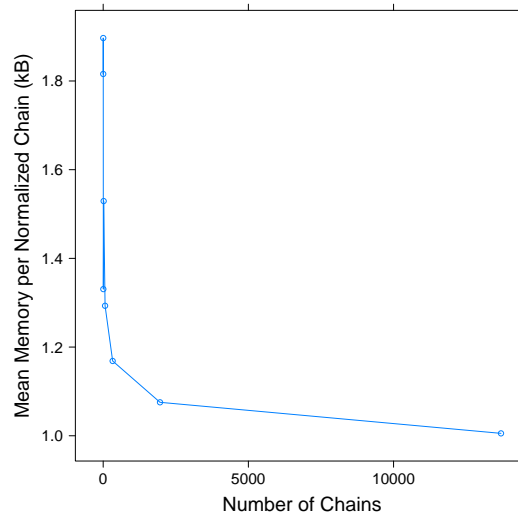
**Table 12.3:** Memory Usage for Mediation

As the process creates a lot of new objects in form of mediator chains and other internally used data structures, it has to be expected that the memory usage increases with the number of chains.



**Figure 12.5:** Memory Usage for the Establishment of Mediator Chains

Figure 12.5 (a) shows that the total memory usage is growing with the number of chains and subfigure (b) shows that the memory usage per chain is growing with the number of chains. However, as expressed by the average chain length in Table 12.3 the number of IROs within a mediator chain is also growing with the number of representations. For four representations, the algorithm results in five mediator chains. Two of these chains contain three IROs, two chains two IROs, and one chain one IRO. Hence, the chains have a length of 2.2 in average. Taking the average length into account while comparing the memory usage of mediator chains and normalizing the chains with the length, it turns out that the memory usage is shrinking with the number of chains (Figure 12.6). The main reason for this is that objects are shared by several chains.



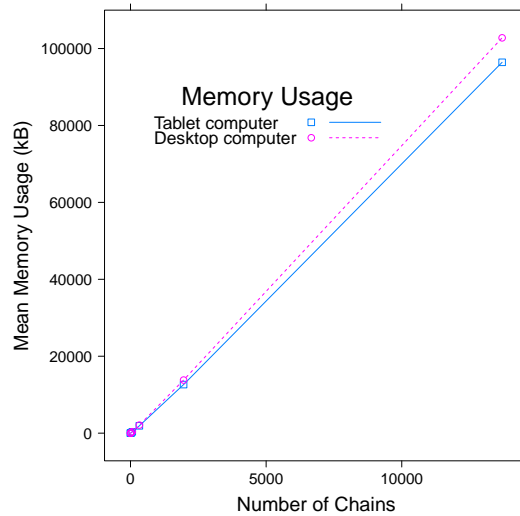
**Figure 12.6:** Mean Memory Usage per Normalized Mediator Chain

We also analysed the memory consumption during the test on the second device. The results of this analysis are shown in Table 12.4.

Representations				Mean (kB)	Standard deviation (kB)
	IROs	Chains	n		
2	2	1	50	2.147	0.013
3	6	2	50	7.047	0.042
4	12	5	50	22.833	0.611
5	20	16	50	83.622	2.764
6	30	65	50	372.624	9.747
7	42	326	50	2095.470	85.993
8	56	1957	50	13807.709	459.081
9	72	13700	50	102787.474	5007.701

**Table 12.4:** Memory Usage for Mediation (Desktop Computer)

The comparison of the memory usage of the two devices (see Figure 12.7) shows that both require nearly the same amount of memory.



**Figure 12.7:** Comparison of the Memory Usage on Different Devices during the Establishment of Mediator Chains

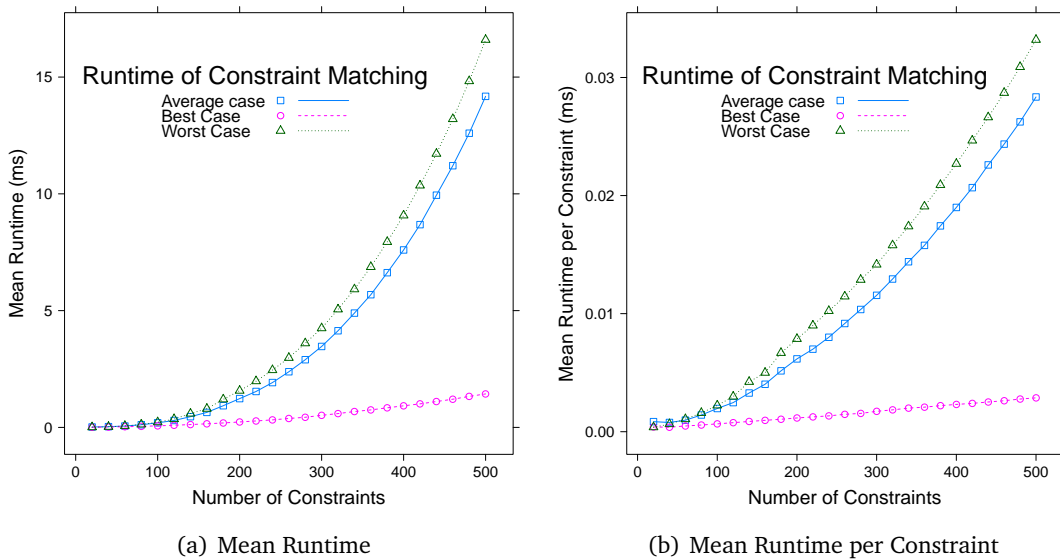
But when comparing the standard deviation of the evaluations on both devices, it turns out that the deviation is significantly higher for the desktop device compared to the Android tablet. This effect is hard to explain, but the most likely reason is the Java garbage collector. As the desktop computer is much faster and has more memory compared to the tablet computer, the garbage collector is called less often during the tests (especially for a high number of established chains). This also explains the increased runtime for nine representations on the tablet computer. The garbage collector is called very often and thus retards the actual process.

## 12.2 Constraint Matching

The next phase for checking a matching of a context offer and a context query is the evaluation of the metadata constraint (see Section 8.3). Metadata constraints are used in both context queries and context offers to precise which information is required respectively offered under which condition. The constraint matching is based on semantic tableaux. The implementation used in this work has been provided by Alexander Kohout as part of this Bachelor thesis [69]. This work has been supervised by the author of this thesis.

For the evaluation we distinguish between three different scenarios: best, average and worst case scenario. Every test has been performed for 20 up to 500 constraints (step size has been 20 constraints). Furthermore, every test has been repeated 20 times to provide a certain statistical persistence. The three different scenarios differ in the way how the constraints are composed.

As correctly described by Kohout, “[...] according to our tableaux algorithm, a best case scenario is given when a logical expression consisting of conjunctions only has to be analyzed. Here the algorithm expands the tableau without ever causing a branch, just adding all components of a conjunction to the end of the tableau’s root node.” [69]. However the last constraint has to contain a contradiction. Otherwise, the tableaux algorithm stops immediately after the start, as no negative matching is possible. “The worst case scenario is given when a logical expression consisting of conjunctions of disjunctions has to be analyzed. According to the general method of tableaux, a disjunction causes a branch while a conjunction just adds all components of a conjunction to the end of the current node. If now only disjunctions with two predicates are defined, and these disjunctions are conjuncted with each other, this will cause the highest possible count of created branches.” [69]. “To create an average case scenario a trade-off between best and worst case scenario has to be found. A kind of average scenario can be simulated by inserting disjunctions within a conjunctive formula, causing  $2k$  branches with  $k$  inserted disjunctions. The here chosen trade-off between conjunctions and disjunctions is a 1 to 3 ratio, i.e. every third conjunction is turned into a disjunction.” [69].



**Figure 12.8:** Scalability of Matching Metadata Constraints.

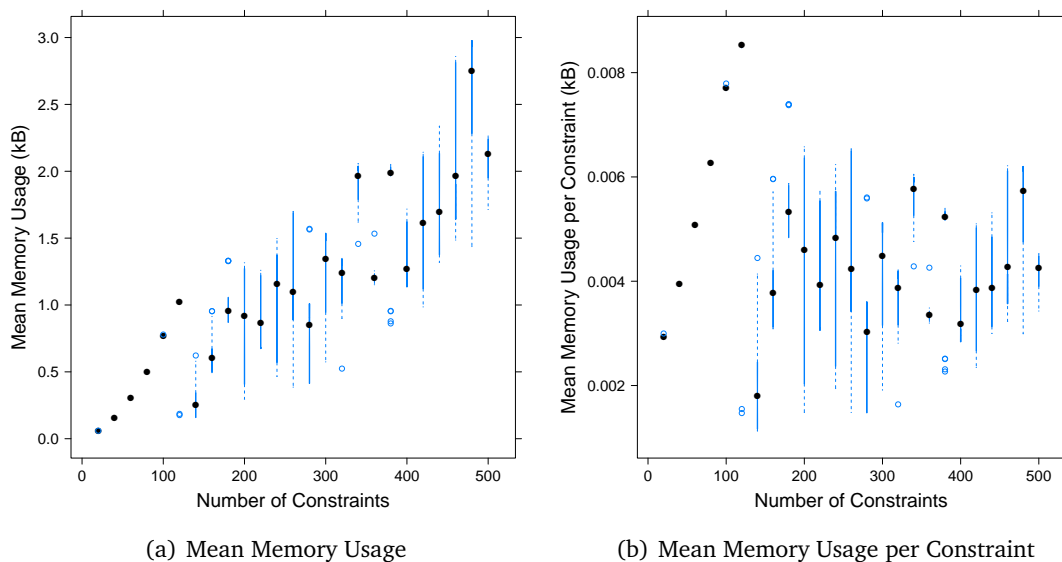
The results of this analysis are visualized in Figure 12.8. As depicted in Figure 12.8 (a) the mean runtime tends to exponential growth in the average and worst case scenarios. From this subfigure, the exact interpretation of the runtime in the best case scenario is not possible: as it is also a curve and not a line, it is at least not a linear dependency. However, after normalizing the runtime with the number of constraints, we see Figure 12.8 (b) that the mean runtime per constraint is growing linearly with the number of constraints in the best case scenario. This means that the constraint matching has a quadratic complexity in the best case scenario. With this result, we also refute the assumption of Kohout, that the constraint matching has a linear complexity in the best case scenario. Anyway, we also increased the number of repetitions and used another device compared to the evaluation of Kohout. The two other scenarios look rather identical considering the curves, which is

a matter of the design of the average case. Nevertheless, the maximal mean runtime in the worst case is 18.39 ms and should be fast enough for our application areas.

Case	Constraints	n	Mean (kB)	Standard deviation (kB)
Average Case	500	20	1.480	0.365
Best Case	500	20	0.895	0.459
Worst Case	500	20	2.083	0.179

**Table 12.5:** Memory Usage for Constraint Matching of 500 Constraints

Table 12.5 summarizes the memory usage while the constraint matching of 500 constraints in all three scenarios. Furthermore Figure 12.9 (a) shows the memory usage in the worst case scenario in dependency to the number of constraints.



**Figure 12.9:** Memory Usage while Matching Metadata Constraints

As depicted in Figure 12.9 (a), the memory usage curve contains many outliers. This can easily be explained. In difference to e.g. the establishment of the mediator chains, the constraint matching does not generate permanently available new objects but rather several internally used but only temporarily available objects. As a consequence and depending on the garbage collector of the Java runtime environment, these temporary objects may already be removed before the actually measurement ends. Figure 12.9 (b) shows the memory usage per constraint in the different test cases as boxplots. The figure again shows that there are many outliers especially in these test cases with a higher number of constraints and hence a higher runtime. However, the maximal usage of 2500 Byte in the worst case means that there is no restriction for this approach for our purposes.



## 12.3 Selection Service

After the matching of context offers and queries, the selection service tries to find an optimal set of mediator chains. This set should satisfy all context requests registered to the system taking into account the different selection criteria of the context queries. It is the selection algorithm described in Section 9.4 that calculates the different sets of mediator chains for which the Aggregated Utility Function (AUF, see Section 9.2.2) is calculated in order to rate the respective set. The set resulting in the highest AUF is the set of mediator chains that is activated.

The AUF is a weighted sum of multiple Single Utility Functions (SUF, see Section 9.2.2) validating the usability of the evaluated set regarding a certain context query and a Cost Minimization Utility Function (CMUF, see Section 9.2.2). As SUF and CMUF only consist of linear combinations of different selection criteria, the calculation of these functions and also of the AUF is of linear complexity. But it strongly depends on the selection algorithm how often the AUF has to be calculated. The number of iterations of the selection algorithm depends on the number of registered context queries and context chains per query.

In Section 9.4 we have already discussed that the complexity of this algorithm is exponential to the number of registered queries. More precisely, the complexity can be estimated by  $(|O_{CS}| + (|CR| \cdot |O_S|)^{|R_{CR}|})^{|R_{CS}|}$ .  $O_{CS}$  is the set of context offers provided by services that do not express any additional context query.  $CR$  is the set of context reasoner, hence of context services that express additional context queries. The set containing all context queries specified by these services is the set  $R_{CR}$ , while  $R_{CS}$  contains any other context request.  $O_S$  is the full set of context offers provided by an arbitrary context service.

In this evaluation, we focus on the complexity from the practical point of view and measure the duration until a selection is done. The selection is finished when all different valid combinations of mediator chains are evaluated. We stepwise increase the number of context queries (starting from one query up to nine queries) and also the number of context offers respectively mediator chains that serve as input for different queries. Every context query specifies two selection criteria: one selection criteria with respect to the accuracy of the requested information and one with respect to the cost. Similarly every offer expresses two metadata criteria regarding accuracy and cost. For every query and every offer in the respective configuration a mediator chain is generated. As a result, for  $n$  queries and  $m$  offers there are  $n \cdot m$  mediator chains. The evaluation for every configuration (1–5 offers and 1–9 queries) is repeated ten times. As we focus in this evaluation on regular services, which do not request for additional context information, the estimation of the complexity can be simplified to  $|O_{CS}|^{|R_{CS}|}$ .

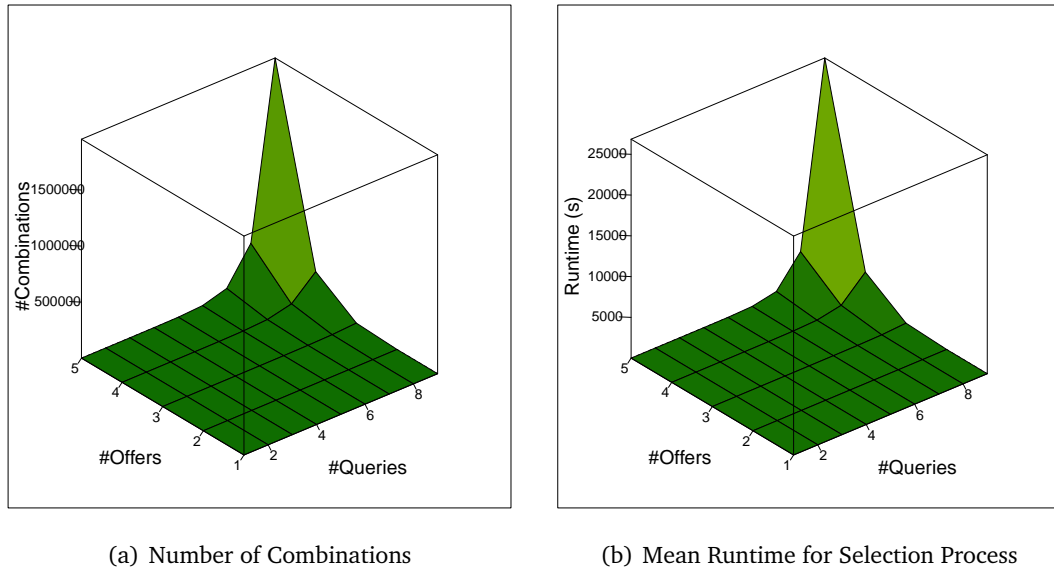
The results of the evaluation are summarized in Table 12.6 and visualized in Figure 12.10.

Queries	Offers	Combinations	n	Mean (ms)	Standard deviation (ms)
1	1	1	10	2.700	0.675
1	2	2	10	3.800	0.422
1	3	3	10	5.200	0.632
1	4	4	10	6.700	0.483

Queries	Offers	Combinations	n	Mean (ms)	Standard deviation (ms)
1	5	5	10	8.700	0.823
2	1	1	10	3.200	0.632
2	2	4	10	8.400	0.699
2	3	9	10	14.400	0.966
2	4	16	10	18.200	0.789
2	5	25	10	26.300	0.823
3	1	1	10	3.100	0.316
3	2	8	10	13.400	0.516
3	3	27	10	41.400	5.337
3	4	64	10	202.000	3.232
3	5	125	10	392.800	23.574
4	1	1	10	3.600	0.516
4	2	16	10	29.600	0.699
4	3	81	10	330.000	8.083
4	4	256	10	1143.300	25.180
4	5	625	10	2813.200	33.999
5	1	1	10	3.900	0.316
5	2	32	10	141.400	6.963
5	3	243	10	1441.500	24.834
5	4	1024	10	6210.300	51.719
5	5	3125	10	19217.800	477.211
6	1	1	10	4.400	0.516
6	2	64	10	466.100	9.983
6	3	729	10	5489.700	184.260
6	4	4096	10	30355.300	184.044
6	5	15625	10	114808.100	237.391
7	1	1	10	6.100	2.132
7	2	128	10	1153.500	18.180
7	3	2187	10	19556.400	102.071
7	4	16384	10	147145.200	252.215
7	5	78125	10	703949.000	3721.804
8	1	1	10	5.900	0.738
8	2	256	10	2770.200	37.832
8	3	6561	10	71885.900	174.011
8	4	65536	10	722746.000	2376.953
8	5	390625	10	4352701.300	11796.179
9	1	1	10	6.600	0.516
9	2	512	10	6796.300	53.112
9	3	19683	10	265032.700	789.166
9	4	262144	10	3551579.800	9863.070
9	5	1953125	10	26852370.800	156932.677

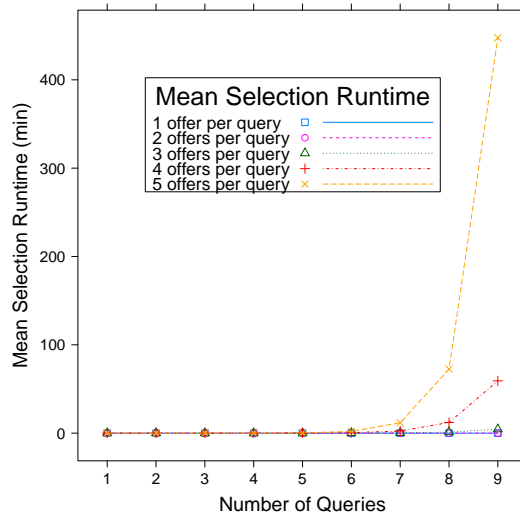
**Table 12.6:** Runtime Statistics

Table 12.6 can be interpreted as following: the column ‘Queries’ contains the number of registered context queries, hence  $|R_{CS}|$ . The column ‘Offers’ comprises the number of context offers that are applicable for a context query. As we evaluate the worst case, we assume that every context offer  $o \in O_{CS}$  is useful for every context query. For that reason, we have in total  $|O_{CS}|^{|R_{CS}|}$  valid combinations of context offers. The number of valid combinations is depicted in the column ‘Combinations’. As described in Section 9.4, we can see that the runtime increases significantly with every additional query. Naturally, the runtime also expands when augmenting the number of offers respectively mediator chains per query.



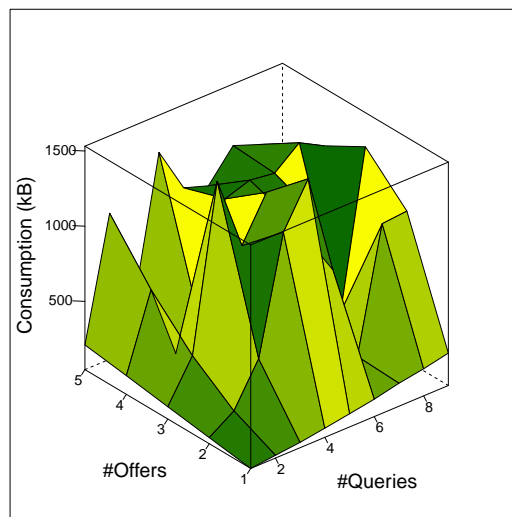
**Figure 12.10:** Mean Runtime of the Selection

More precisely, the runtime grows exponentially with the number of context queries and polynomially with the number of context offers per context query. The polynomial dependency to context offers is not that worse as it is rather improbable that there are so many chains for every query. Also in the unlike case, it would be simple to establish a prefiltering to lower the number of chains per query. Such a prefiltering could be based on the single utility function (SUF) e.g. by calculating the SUF for every chain and choosing only a few of these chains which resulted in a high SUF for the actual selection process. Figure 12.10 (a) shows the number of valid combinations based on the number of context queries and the number of context offers per query, while Figure 12.10 (b) shows the mean runtime for the selection process. Figure 12.11 visualizes the mean runtime of the selection process whereby the results are grouped by the number of offers respectively chains per queries. Consequently, the figure contains five curves. This diagram shows that there is an exponential dependency regarding the number of queries and only a polynomial dependency of the selection runtime with regard to the number of context offers per query. This polynomial dependency is also the main argument to limit the number of results of the mediation check evaluated in the first section of this chapter.



**Figure 12.11:** Mean Runtime of the Selection Grouped by Number of Offers

Figure 12.12 shows the memory usage of the selection process for the different evaluated combinations of queries and offers. From the visualization it becomes clear, that it is not possible to provide clear statements regarding the memory consumption. The selection process only creates temporarily available objects to calculate the different utility functions. Due to the partially long runtime of the process, the garbage collection deletes regularly the unused objects meanwhile. For this reason, we omitted the detailed analysis of the memory consumption. Nevertheless, the selection process is not very memory consuming and uses less than 1500 kB (see Figure 12.12).



**Figure 12.12:** Mean Memory Usage of the Selection Process

# 13 Conclusions

---

*I may not have gone where I intended to go,  
but I think I have ended up where I needed to be.*

– Douglas Adams (1952-2001)  
The Long Dark Tea-Time of the Soul (1988)

This chapter concludes the thesis by summarizing its research contributions and by providing a discussion of key topics for future work.

## 13.1 Summary of Contributions

Nowadays mobile devices such as smartphones or tablet computers are equipped with fast processors, a lot of memory, and a wide range of sensors like accelerometer, GPS sensor, or light sensor. But in the vision of ubiquitous computing devices disappear from the focus of the users. This change of shifting of computers away from direct computer interaction requires another way of applications to interact without bothering the user. Context awareness is a central aspect of the solution for this problem.

Different information providers can serve as input provider for context-aware applications. In recent years, a lot of attention has been drawn to the development of reasoning mechanisms. These mechanisms are able to retrieve high-level information from other information, which can be offered by these sensors or by external information providers like databases, information system or external sensors. Context-aware applications use the information provided by sensors, reasoning mechanisms and other sources for visualization purposes: Beyond that, context-aware self-adaptive applications are able to adjust their behaviour without the intrusion of the user based on this information.

The ongoing trend of ubiquitous computing brings several new challenges, which have also to be taken into account during the development of context-aware applications in these environments. For example, it is not possible for the developer of a context-aware application to know all possibly available information providers in the environment during the development phase. For this reason developers should rather explicitly describe which kind of information they expect instead of using an explicit source. On the other hand, this requires also information sources to offer an concrete description of the information they can provide. The abstraction of all different kinds of information providers to context services allows for transparently discover new information sources and to access the provided information by a common well known interface.

Abstracting information providers as context services comes along with several requirements on the underlying system. A central requirement is the loose coupling of information providers and information consumers. In our system, methods for information access and retrieval are encapsulated in context services implementing the common interface. The context services are dynamically bound to the system if they can serve as input provider for one or more information consumers.

Context information can be represented in several ways. But only providing a common interface to access the information from heterogeneous sources is not sufficient to account the heterogeneous representations of context information and of its metadata. For that reason, we developed a context model allowing the explicit definition of the semantics and data structures of the information. This model has initially been inspired by the CoOL approach developed by Strang et al. [134]. But CoOL only focused on the actual information but not on metadata. The information on semantics and how the data are structured are stored centrally in a context ontology. This ontology consists of several parts. The top-level ontology contains the most important concepts, classes and relationships while it is extensible by domain- and application-specific ontologies.

The requirement for loose coupling is attended by several more requirements, which have to be fulfilled to offer an appropriate support for context services. Context services should be dynamically discoverable as it is not possible to know all providers at design time. As a consequence, our middleware contains a discovery service which is extensible by discovery plug-ins. The discovery service itself only discovers locally available context services. The discovery mechanisms for external discovery using different discovery protocols are sourced out to the discovery plug-ins.

The dynamic discovery and the transparent access by the context-aware applications require for a language to describe which kind of information are requested or offered. As it is also not possible to know in which form the information is offered (data structures, semantics, metadata, etc.), the system and consequently also the language needs to be able to describe and handle heterogeneously represented information. For this purpose we developed the Context Offer and Query Language (COQL). The COQL builds on the context model and refers to the different classes and individuals of the context ontology to specify which information is provided or required. Both context offers and context queries can include constraints to further precise the offered or requested information.

Based on these descriptions, a matching process finds appropriate context services for an information consumer. This matching process consists of a mediation phase and a constraint matching. Along with our context model, we adopted the concept of Inter-Representation Operations (IROs) from the CoOL project. IROs are able to transfer information from one representation into another representation. In addition to this concept, we developed the more generic concept of metadata operators, which allow the calculation of new metadata based on other existing metadata. In the mediation phase, the system establishes chains of IROs and metadata operators in order to transfer the offered information into the requested information. This mediation also includes the metadata and their representations. After the generation of at least one mediator chain, the constraints of context offer and context query are compared to prove their satisfaction. We applied the method of analytic tableaux for the prove of satisfaction.

While it is possible that some information providers offer the same type of information (e.g. information regarding the current position or the current activity), this information

can differ in quality levels and also in costs (energy consumption, resource usage, or monetary costs) for providing and calculating the information. This requires the selection process to consider the requirements of the information consumer to find the best service. Another challenge for such a system is the reduction of energy consumption. Also modern mobile devices have a limited energy capacity. It is desirable to reduce the energy consumption as much as possible. This goes along with two different requirements on the system: first, services that are not selected as an information provider should be deactivated to save energy and second it is possible to share a service among more than one consumer. To facilitate this support, the system favours the selection and activation of these services that can be used by more than one consumer. Both energy related requirements highly influence the selection process. In our approach we use utility functions to calculate the usefulness of a context offer serving as an input provider for a context query. In difference to other systems, our system searches for a whole set of context offers that may serve as input providers for all context queries. The selection process has to incorporate historical context information and its metadata. Due to the deactivation of unused context services, the process cannot assume to get up-to-date information.

Our system stands out from the existing works by several aspects. We significantly improved the context model by supporting metadata and more generic operations on the information, even if the model is based on CoOL [134] and the revised version by Reichle [109]. The COQL, which builds on the Context Query Language [158] and its successor (the Information Offer and Request Language) [109], has been extended by support for heterogeneously represented metadata and for expressing selection preferences. The matching process also stands out. Firstly, the mediation process as part of the matching process automatically establishes mediator chains. Other approaches like the CoCo system by Buchholz et al. allow also to mediate between different representations of information but require a manual configuration of the required meditations. Secondly, we apply the method of analytic tableaux in the matching process to prove the satisfiability of the constraints. The usage of the method of analytic tableaux to solve constraint satisfaction problems or for ontology reasoning is described in several existing works. Nevertheless, the usage in the domain of context-aware applications on a resource-limited device is not common and outstanding. The selection approach also has to be highlighted. While an utility-based selection is already been employed in other projects like CONTEXT [18], to the best of our knowledge, none of the existing works tries to select a whole set of context sources to support the sharing of information and to minimize the resource consumption. The aspect of resource consumption is another highlight of our work. The idea of activation respectively deactivation of context sources has been adapted from Paspallis [104]. However, the system by Paspallis did not provide a complex selection mechanism. Furthermore, we discussed the problems that come along with the activation respectively deactivation and provided initial solutions for the selection on historical data.

In order to prove and to demonstrate the developed concepts, a context middleware, a set of small demonstrators, and performance and scalability tests have been developed. While the demonstrators show the coverage of the different requirements, the scalability and performance tests also point out the disadvantages of our solutions. Especially the scalability of the selection approach is improvable due to its exponential complexity depending on the number of requests. Nevertheless, we have demonstrated that our

approach is useful even for a large set of offers and queries (our largest demonstrator consists of 23 context services offering 31 different context offers and 6 context consumers). The battery runtime evaluation has to be emphasized as it shows the high potentials of our approach with respect to minimize resource consumption.

## 13.2 Outlook and Future Work

Although the wide range of contributions of this work in the areas of context-aware computing especially with the focus on discovery, matching and selection of heterogeneously represented context information and their providers, our work has of course several open issues which remain as future work:

- **Combined selection and fusion approach:** several existing works (like Nexus [9, 112, 97, 43, 79, 58] or Reichle [109]) fuse context information either to increase reliability of the provided data (competitive fusion) or to infer new context information (complementary fusion). While it is already possible to develop context services encapsulating complementary fusion mechanisms with the current system, the support for competitive fusion is missing. This would allow to select more than one context service for a request.
- **Enhance selection scalability:** Currently the selection approach as described in Chapter 9 selects one or more offers at once in order to satisfy one or more context requests. Even if several enhancements are already implemented (see Section 9.4), the selection algorithm can be further improved. The context selection is performed once for all requests in order to share offers and to reduce cost. It might be sufficient to divide the selection process into several sub-processes. Every sub-process selects mediator chains (and as a consequence context offers) only for these requests, which have at least one chain which use the same context offer as input provider. Theoretically this should result in the same selection set as by the implemented selection algorithm but should reduce the number of calculations. Additionally, it would be possible to evaluate the usability of common optimization methods.
- **Improved calculation of mediator chains:** In the matching and mediation phase, as described in Chapter 8, mediator chains are established to transfer information from an offered representation into the requested format. A mediator chain is a sequence of one or more Inter-Representation Operations and of metadata operations. To calculate these mediator chains, the current algorithms stops either if no more chains can be computed or if a maximal number of chains is reached. This maximum is specified as a system parameter. As shown in the evaluation of the selection function it is beneficial to limit the number of potential context providers per query. For this purpose, the usability of optimization algorithms like the Dijkstra algorithm has to be researched. But these algorithms cannot be simply used due the complex transformation of the metadata.
- **Development methodology:** Introducing context services as transparently accessible reusable context providers change the way how to develop context-aware applications. Instead of focussing on methods for accessing the required information from the underlying hardware and for reasoning on these information to derive new information, the application developer has to know how to



use constraints and selection function factors in order to retrieve the required information. Several methodologies exist (like the methodology by Henriksen et al. [51]) that already rely on approaches encapsulating context providers in separate reusable components. None of these approaches allow the expression of complex queries (including constraints and selection preferences) to transparently discover these providers.

- **Enhanced support for retrieving metadata:** Metadata are a central aspect in this work as they are used to precise context queries and offers to construct appropriate mediator chains and for the selection of context providers. As we focused in this work on providing a generic support, we reused in the demonstrators either statements regarding metadata raised in existing works or we forged values only for demonstration purposes. Hence, in future work the middleware could provide support for developers of context services to measure or learn different metadata of their context services and to be able to specify correct metadata constraints.
- **Remote context services:** The remote binding of context services is already possible within this work, but we currently do not support the revision of remote context offers. This is actually required e.g. to update required cost (in general this will be communication cost and not cost caused by a sensor). On top of that, this support is also necessary for updating quality related metadata, e.g. metadata with the scope 'freshness' have to be recalculated after transmission as they are not valid any more.
- **Privacy and security:** The topics of security and privacy have not been taken into account within this work. For these topics a lot of additional questions have to be answered: How to ensure a secure transmission of context data only to foreseen receivers? How to share context data only to selected devices or persons? How to prevent intrusion into and manipulation of context services respectively context data? The solutions for these problems would intrude all parts of this work and would also influence a potential development methodology.
- **Extended constraint support:** Currently only unary constraints and their combinations are supported in both context offers and queries to precise them. In the future, the COQL should be extended with regard to allow for more complex constraints. With n-ary constraints it would be possible to compare variables or also to define complex expressions like *distanceBetween(friend.position, my.position) < 1km*, which could be used e.g. as a scope constraint.



**Part IV**  
**Appendices**



## A Logging Output for Demonstrator B

---

Listing A.1 shows the logging output of the selection service while calculating the optimal selection for the second query of Demonstrator B (see Section 11.4).

```
1 Start selection process
2 #Queries=1
3 #QueriesForReasoner=3
4 #Chains=9
5 #AverageChainsPerQueries=2
6 #Combinations = 6
7 Checking combination #1 of 6
8 calculateSingleUtilityFunction for query =query_activity_demo2-1343829250916-30
9 No selection function specified → utility = 0.5
10 calculateSingleUtilityFunction for query =query_ActivityKose_ACCELERATION_1
    -1343829250124-60
11 No selection function specified → utility = 0.5
12 calculateAggregatedUtilityFunction for set of chains (
    m_offer_ANDROID_ACCELEROMETER_Generic_1-1343829249900-33
    _query_ActivityKose_ACCELERATION_1-1343829250124-60-1343829258283-87;
    m_offer_ActivityKose-1343829250124-51_query_activity_demo2
    -1343829250916-30-1343829258564-19;) = 0.9500000005
13 Checking combination #2 of 6
14 calculateSingleUtilityFunction for query =query_activity_demo2-1343829250916-30
15 No selection function specified → utility = 0.5
16 calculateSingleUtilityFunction for query =query_ActivityKose_ACCELERATION_1
    -1343829250124-60
17 No selection function specified → utility = 0.5
18 calculateAggregatedUtilityFunction for set of chains (
    m_offer_ANDROID_ACCELEROMETER_Generic_2-1343829249900-37
    _query_ActivityKose_ACCELERATION_1-1343829250124-60-1343829258345-9;
    m_offer_ActivityKose-1343829250124-51_query_activity_demo2
    -1343829250916-30-1343829258564-19;) = 0.9500000005
19 Checking combination #3 of 6
20 calculateSingleUtilityFunction for query =query_activity_demo2-1343829250916-30
21 No selection function specified → utility = 0.5
22 calculateSingleUtilityFunction for query =query_ActivityKwapisc_ACCELERATION
    -1343829250132-45
23 No selection function specified → utility = 0.5
24 calculateAggregatedUtilityFunction for set of chains (
    m_offer_ANDROID_ACCELEROMETER_Generic_2-1343829249900-37
    _query_ActivityKwapisc_ACCELERATION-1343829250132-45-1343829258360-60;
    m_offer_ActivityKwapisc-1343829250132-5_query_activity_demo2
    -1343829250916-30-1343829258588-44;) = 0.9500000005
25 Checking combination #4 of 6
26 calculateSingleUtilityFunction for query =query_activity_demo2-1343829250916-30
27 No selection function specified → utility = 0.5
28 calculateSingleUtilityFunction for query =query_ActivityKwapisc_ACCELERATION
    -1343829250132-45
29 No selection function specified → utility = 0.5
30 calculateAggregatedUtilityFunction for set of chains (m_offer_ActivityKwapisc
    -1343829250132-5_query_activity_demo2-1343829250916-30-1343829258588-44;
    m_offer_ANDROID_ACCELEROMETER_Generic_1-1343829249900-33
    _query_ActivityKwapisc_ACCELERATION-1343829250132-45-1343829258305-48;) =
    0.9500000005
31 Checking combination #5 of 6
32 calculateSingleUtilityFunction for query =query_activity_demo2-1343829250916-30
33 No selection function specified → utility = 0.5
```

```

34 calculateSingleUtilityFunction for query =query_ActivityLee_ACCELERATION
    -1343829250178-87
35 No selection function specified --> utility = 0.5
36 calculateAggregatedUtilityFunction for set of chains (
    m_offer_ANDROID_ACCELEROMETER_Generic_1-1343829249900-33
    _query_ActivityLee_ACCELERATION-1343829250178-87-1343829258314-90;
    m_offer_ActivityLee-1343829250178-66_query_activity_demo2
    -1343829250916-30-1343829258613-77;) = 0.9500000005
37 Checking combination #6 of 6
38 calculateSingleUtilityFunction for query =query_activity_demo2-1343829250916-30
39 No selection function specified --> utility = 0.5
40 calculateSingleUtilityFunction for query =query_ActivityLee_ACCELERATION
    -1343829250178-87
41 No selection function specified --> utility = 0.5
42 calculateAggregatedUtilityFunction for set of chains (
    m_offer_ANDROID_ACCELEROMETER_Generic_2-1343829249900-37
    _query_ActivityLee_ACCELERATION-1343829250178-87-1343829258368-0;
    m_offer_ActivityLee-1343829250178-66_query_activity_demo2
    -1343829250916-30-1343829258613-77;) = 0.9500000005
43
44 Selection finished in 33ms: Selected chains = {Chain(Offer=
    offer_ANDROID_ACCELEROMETER_Generic_1-1343829249900-33;Query=
    query_ActivityKose_ACCELERATION_1-1343829250124-60):Acceleration|->
    AndroidDeviceAccelerationRep , Chain(Offer=offer_ActivityKose
    -1343829250124-51;Query=query_activity_demo2-1343829250916-30):Activity|->
    ActivityKoseRep ,} utilityOfSelectionSet = 0.9500000005
45

```

**Listing A.1:** Demonstrator B: Logging Output of the Selection Service for the Second Scenario

## B Erklärung

---

Hiermit versichere ich, dass ich die vorliegende Dissertation selbstständig, ohne unerlaubte Hilfe Dritter angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Dritte waren an der inhaltlich-materiellen Erstellung der Dissertation nicht beteiligt; insbesondere habe ich hierfür nicht die Hilfe eines Promotionsberaters in Anspruch genommen. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren verwendet worden.

Kassel, im November 2012

---

Dipl.-Inf. Michael Wagner





# C Bibliographies

---

## C.1 Bibliography

- [1] Zied Abid, Sophie Chabridon and Denis Conan. **A Framework for Quality of Context Management**. In: *Quality of Context, First International Workshop, QuaCon 2009, Stuttgart, Germany, June 25-26, 2009. Revised Papers*. Ed. by Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger and Frank Dürr. Vol. 5786. LNCS. Springer, 2009, pp. 120–131. isbn: 978-3-642-04558-5 (cit. on pp. 50–51, 60, 150).
- [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, First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*. Ed. by Hans-Werner Gellersen. Vol. 1707. LNCS. Springer, 1999, pp. 304–307. isbn: 3-540-66550-1 (cit. on pp. 16–17, 43–44).
- [3] Alessandra Agostini, Claudio Bettini, Nicolò Cesa-Bianchi, Dario Maggiorini, Daniele Riboni, Michele Ruberl, Cristiano Sala and Davide Vitali. **Towards Highly Adaptive Services for Mobile Computing**. In: *Mobile Information Systems, IFIP TC 8 Working Conference on Mobile Information Systems (MOBIS), 15-17 September 2004, Oslo, Norway*. Ed. by Elaine Lawrence, Barbara Pernici and John Krogstie. Vol. 158. IFIP International Federation for Information Processing. 2004, pp. 121–134. isbn: 0-387-22851-9 (cit. on pp. 47, 60).
- [4] Alessandra Agostini, Claudio Bettini and Daniele Riboni. **A Performance Evaluation of Ontology-Based Context Reasoning**. In: *Fifth Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2007), 19-23 March 2007, White Plains, New York, USA*. IEEE Computer Society, 2007, pp. 3–8. isbn: 978-0-7695-2788-8 (cit. on pp. 47, 60, 149).
- [5] Alessandra Agostini, Claudio Bettini and Daniele Riboni. **Loosely Coupling Ontological Reasoning with an Efficient Middleware for Context-awareness**. In: *2nd Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2005), 17-21 July 2005, San Diego, CA, USA*. IEEE Computer Society, 2005, pp. 175–182. isbn: 0-7695-2375-7 (cit. on pp. 47, 60).
- [6] Marco Aiello, Ganna Frankova and Daniela Malfatti. **What's in an Agreement? An Analysis and an Extension of WS-Agreement**. In: *Service-Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005, Proceedings*. Ed. by Boualem Benatallah, Fabio Casati

- and Paolo Traverso. Vol. 3826. LNCS. Springer, 2005, pp. 424–436. isbn: 3-540-30817-2 (cit. on p. 92).
- [7] Grigoris Antoniou and Frank van Harmelen. **Web Ontology Language: OWL**. In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Second Edition. International Handbooks on Information Systems. Springer, 2009, pp. 67–92. isbn: 978-3-540-70999-2 (cit. on p. 39).
- [8] Matthias Baldauf, Schahram Dustdar and Florian Rosenberg. **A survey on context-aware systems**. In: *IJAHUC 2.4 (2007)*, pp. 263–277 (cit. on pp. 17, 43–44, 57).
- [9] Christian Becker. **System Support for Context-aware Computing**. Englisch. Habilitation. Universität Stuttgart : Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Germany, June 2004, p. 245. url: <http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTR/L/NCSTR/view.pl?id=HABIL-2004-02&engl=> (cit. on pp. 4–5, 56, 61, 90, 192).
- [10] Christian Becker and Frank Dürr. **On location models for ubiquitous computing**. In: *Personal and Ubiquitous Computing 9.1 (2005)*, pp. 20–31 (cit. on pp. 4, 90).
- [11] Christian Becker and Daniela Nicklas. **Where do spatial context-models end and where do ontologies start? A proposal of a combined approach**. Deutsch. In: *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management*. Ed. by Jadwiga Indulska and David De Roure. Nottingham, England, Sept. 2004, pp. 48–53. isbn: 854328130 (cit. on pp. 16–17).
- [12] Evert Willem Beth. **The Foundations of Mathematics**. Amsterdam, North-Holland Pub. Co., 1959 (cit. on p. 28).
- [13] Gregory Biegel and Vinny Cahill. **A Framework for Developing Mobile, Context-aware Applications**. In: *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA*. IEEE Computer Society, 2004, pp. 361–365. isbn: 0-7695-2090-1 (cit. on pp. 44, 57, 61).
- [14] Steffen Bleul and Kurt Geihs. **Automatic Quality-Aware Service Discovery and Matching**. In: *Proceedings of the 13th Annual Workshop of HP OpenView University Association (HP-OVUA)*. Infonomics-Consulting, Stuttgart, Germany, May 2006, pp. 109–118. isbn: 3-000-18780-4 (cit. on p. 92).
- [15] Steve Bratt. **Semantic Web, and Other Technologies to Watch**. Presentation. 2007. url: <http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/> (cit. on p. 40).
- [16] Thomas Buchholz, Michael Krause, Claudia Linnhoff-Popien and Michael Schiffers. **CoCo: Dynamic Composition of Context Information**. In: *1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, 22-25 August 2004, Cambridge, MA, USA*. IEEE Computer Society, 2004, pp. 335–343. isbn: 0-7695-2208-4 (cit. on pp. 46, 60, 62, 110, 191).

- [17] Thomas Buchholz, Axel Küpper and Michael Schiffers. **Quality of Context Information: What it is and why we need it.** In: *Proceedings of the 10th HP-OVUA Workshop, 2003, Geneva, Switzerland*. July 2003 (cit. on pp. 18–22, 46–47).
- [18] Maria Chantzara, Miltiades E. Anagnostou and Efstathios D. Sykas. **Designing a Quality-Aware Discovery Mechanism for Acquiring Context Information.** In: *20th International Conference on Advanced Information Networking and Applications (AINA 2006), 18-20 April 2006, Vienna, Austria*. IEEE Computer Society, 2006, pp. 211–216. isbn: 0-7695-2466-4 (cit. on pp. 48–49, 60, 62, 191).
- [19] Harry Chen. **An Intelligent Broker Architecture for Pervasive Context-Aware Systems.** PhD thesis. University of Maryland, Baltimore Count, Dec. 2004. url: <http://ebiquity.umbc.edu/get/a/publication/152.pdf> (cit. on pp. 44, 48, 60).
- [20] Harry Chen, Tim Finin and Anupam Joshi. **A Context Broker for Building Smart Meeting Rooms.** In: *AAAI 2004 Spring Symposium on Knowledge Representation and Ontology for Autonomous Systems*. Draft. Stanford, 2004. url: <http://www.aaai.org/Papers/Symposia/Spring/2004/SS-04-04/SS04-04-008.pdf> (cit. on pp. 17, 48, 60).
- [21] Harry Chen, Tim Finin and Anupam Joshi. **An Ontology for Context-Aware Pervasive Computing Environments.** In: *The Knowledge Engineering Review* 18.03 (2003), pp. 197–207. doi: DOI : 10 . 1017 / S0269888904000025. eprint: [http://journals.cambridge.org/article\\_S0269888904000025](http://journals.cambridge.org/article_S0269888904000025). url: <http://dx.doi.org/10.1017/S0269888904000025> (cit. on pp. 44, 48, 60).
- [22] Harry Chen, Timothy W. Finin and Anupam Joshi. **Using OWL in a Pervasive Computing Broker.** In: *Proceedings of the Workshop on Ontologies in Agent Systems (OAS 2003), Melbourne, Australia, July 15, 2003*. Ed. by Stephen Cranefield, Timothy W. Finin, Valentina A. M. Tamma and Steven Willmott. Vol. 73. CEUR Workshop Proceedings. CEUR-WS.org, 2003, pp. 9–16 (cit. on pp. 48, 60).
- [23] Alonzo Church. **A Note on the Entscheidungsproblem.** In: *J. Symb. Log.* 1.1 (1936), pp. 40–41 (cit. on p. 101).
- [24] Ed Clarke and Andrei Voronkov (Programme Committee Chairs). **16th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR-16).** In: *Call for Paper*. 2010. url: <http://www.lpar.net/lpar-16/cfp.cgi> (cit. on p. 27).
- [25] Yann Collette and Patrick Siarry. **Multiobjective optimization: principles and case studies.** Decision engineering. Springer, 2003. isbn: 3-540-40182-2. url: <http://books.google.de/books?id=XNYF4h1toF0C> (cit. on pp. 113, 136).
- [26] Michael Compton, Corey Henson, Holger Neuhaus, Laurent Lefort and Amit Sheth. **A Survey of the Semantic Specification of Sensors.** In: *2nd International Workshop on Semantic Sensor Networks, at 8th International Semantic Web Conference*. Washington DC, USA, 2009th Oct. 2009 (cit. on p. 90).

- [27] Denis Conan, Romain Rouvoy and Lionel Seinturier. **Scalable Processing of Context Information with COSMOS**. In: *Distributed Applications and Interoperable Systems, 7th IFIP WG 6.1 International Conference, DAIS 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings*. Ed. by Jadwiga Indulska and Kerry Raymond. Vol. 4531. LNCS. Springer, 2007, pp. 210–224. isbn: 978-3-540-72881-8 (cit. on pp. 50–51, 60).
- [28] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider and Lynn Andrea Stein. **DAML+OIL Reference Description**. W3C Note, Dec. 2001. url: <http://www.w3.org/TR/daml+oil-reference> (cit. on p. 39).
- [29] Jiangpeng Dai, Xiaole Bai, Zhimin Yang, Zhaohui Shen and Dong Xuan. **Mobile phone-based pervasive fall detection**. In: *Personal and Ubiquitous Computing* 14.7 (2010), pp. 633–643 (cit. on p. 149).
- [30] Jiangpeng Dai, Xiaole Bai, Zhimin Yang, Zhaohui Shen and Dong Xuan. **PerFallD: A pervasive fall detection system using mobile phones**. In: *Eighth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2010, March 29 - April 2, 2010, Mannheim, Germany, Workshop Proceedings*. IEEE, 2010, pp. 292–297 (cit. on p. 149).
- [31] Indraneel Das and J. E. Dennis. **Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems**. In: *SIAM J. on Optimization* 8 (3 Mar. 1998), pp. 631–657. issn: 1052-6234. doi: 10.1137/S1052623496307510. url: <http://dl.acm.org/citation.cfm?id=588907.589322> (cit. on p. 137).
- [32] Anind K. Dey. **Providing architectural support for building context-aware applications**. AAI9994400. PhD thesis. Atlanta, GA, USA: Georgia Institute of Technology, 2000. isbn: 0-493-01246-X (cit. on pp. 49, 60).
- [33] Anind K. Dey and Gregory D. Abowd. **The Context Toolkit: Aiding the Development of Context-Aware Applications**. In: *Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland*. June 2000 (cit. on pp. 44, 49–50, 60).
- [34] Anind K. Dey, Jennifer Mankoff, Gregory D. Abowd and Scott Carter. **Distributed mediation of ambiguous context in aware environments**. In: *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology, Paris, France, October 27-30, 2002*. Ed. by Michel Beaudouin-Lafon. ACM, 2002, pp. 121–130. isbn: 1-58113-488-6 (cit. on p. 17).
- [35] Li Ding, Pranam Kolari, Zhongli Ding and Sasikanth Avancha. **Using Ontologies in the Semantic Web: A Survey**. In: *Ontologies*. Ed. by Raj Sharman, Rajiv Kishore and Ram Ramesh. Vol. 14. Integrated Series in Information Systems. Springer US, 2007, pp. 79–113. isbn: 978-0-387-37022-4 (cit. on pp. 38–39).
- [36] David S. Doermann and Ramani Duraiswami, eds. **Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, MUM 2004, College Park, Maryland, USA, October 27-29, 2004**. Vol. 83. ACM

- International Conference Proceeding Series. ACM, 2004. isbn: 1-58113-981-0.
- [37] Robert T. Eckenrode. **Weighting Multiple Criteria**. In: *Management Science* 12.3 (1965), pp. 180–192. doi: 10.1287/mnsc.12.3.180. eprint: <http://mansci.journal.informs.org/content/12/3/180.full.pdf+html>. url: <http://mansci.journal.informs.org/content/12/3/180.abstract> (cit. on p. 137).
- [38] Thomas Erl. **SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007. isbn: 0132344823 (cit. on pp. 5, 18).
- [39] **Fifth Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2007), 19-23 March 2007, White Plains, New York, USA**. IEEE Computer Society, 2007. isbn: 978-0-7695-2788-8.
- [40] Patrik Floréen, Michael Przybilski, Petteri Nurmi, Johan Koolwaaij, Anthony Tarlano, Matthias Wagner, Marko Luther, Fabien Bataille, Mathieu Bousard, Bernd Mrohs and Sianlun Lau. **Towards a Context Management Framework for MobiLife**. In: *In IST Mobile & Wireless Communications Summit*. 2005 (cit. on pp. 55, 61–62).
- [41] Kurt Geihs. **Selbst-adaptive Software**. In: *Informatik Spektrum* 31.2 (2008), pp. 133–145 (cit. on pp. 3, 24–25).
- [42] Kurt Geihs et al. **A comprehensive solution for application-level adaptation**. In: *Softw., Pract. Exper.* 39.4 (2009), pp. 385–422 (cit. on p. 25).
- [43] Matthias Großmann, Nicola Höhle, Carlos Lübke and Harald Weinschrott. **An Abstract Processing Model for the Quality of Context Data**. In: *Quality of Context, First International Workshop, QuaCon 2009, Stuttgart, Germany, June 25-26, 2009. Revised Papers*. Ed. by Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger and Frank Dürr. Vol. 5786. LNCS. Springer, 2009, pp. 132–143. isbn: 978-3-642-04558-5 (cit. on pp. 56, 61, 192).
- [44] Thomas R. Gruber. **A translation approach to portable ontology specifications**. In: *Knowl. Acquis.* 5.2 (June 1993), pp. 199–220. issn: 1042-8143. doi: 10.1006/knac.1993.1008. url: <http://dx.doi.org/10.1006/knac.1993.1008> (cit. on pp. 38–39).
- [45] Nicola Guarino, Daniel Oberle and Steffen Staab. **What Is an Ontology?** In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Second Edition. International Handbooks on Information Systems. Springer, 2009, pp. 1–17. isbn: 978-3-540-70999-2 (cit. on p. 38).
- [46] Tao Gu, Hung Keng Pung and Da Qing Zhang. **A Middleware for Building Context-Aware Mobile Services**. In: *In Proceedings of IEEE Vehicular Technology Conference (VTC)*. 2004 (cit. on pp. 44, 58, 61, 79).

- [47] Tao Gu, Hung Keng Pung and Daqing Zhang. **A service-oriented middleware for building context-aware services**. In: *J. Network and Computer Applications* 28.1 (2005), pp. 1–18 (cit. on pp. 58, 61).
- [48] Tao Gu, Xiaohang Wang, Hung Keng Pung and Daqing Zhang. **An Ontology-based Context Model in Intelligent Environments**. In: *Communication Networks and Distributed Systems Modeling and Simulation Conference*. San Diego, California, Jan. 2004 (cit. on pp. 58, 61).
- [49] Jacek Gwizdka. **What's in the context?** In: *Workshop on the What, Who, Where, When, and How of Context-Awareness*. 2000 (cit. on p. 16).
- [50] P. E. Hart, N. J. Nilsson and B. Raphael. **A Formal Basis for the Heuristic Determination of Minimum Cost Paths**. In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), pp. 100–107. doi: <http://dx.doi.org/10.1109/TSSC.1968.300136>. url: <http://dx.doi.org/10.1109/TSSC.1968.300136> (cit. on p. 137).
- [51] Karen Henriksen and Jadwiga Indulska. **Developing context-aware pervasive computing applications: Models and approach**. In: *Pervasive and Mobile Computing* 2.1 (2006), pp. 37–64 (cit. on p. 193).
- [52] Karen Henriksen, Jadwiga Indulska and Andry Rakotonirainy. **Infrastructure for Pervasive Computing: Challenges**. In: *GI Jahrestagung (1)*. 2001, pp. 214–222 (cit. on pp. 14–15, 24).
- [53] Karen Henriksen, Jadwiga Indulska and Andry Rakotonirainy. **Modeling Context Information in Pervasive Computing Systems**. In: *Pervasive Computing, First International Conference, Pervasive 2002, Zürich, Switzerland, August 26–28, 2002, Proceedings*. Ed. by Friedemann Mattern and Mahmoud Naghshineh. Vol. 2414. LNCS. Springer, 2002, pp. 167–180. isbn: 3-540-44060-7 (cit. on p. 17).
- [54] Colombe Hérault, Gaël Thomas and Philippe Lalanda. **A distributed service-oriented mediation tool**. In: *2007 IEEE International Conference on Services Computing (SCC 2007), 9–13 July 2007, Salt Lake City, Utah, USA*. IEEE Computer Society, 2007, pp. 403–409 (cit. on p. 114).
- [55] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph and York Sure. **Semantic Web: Grundlagen**. Berlin: Springer, 2008. isbn: 978-3-540-33993-9. doi: 10.1007/978-3-540-33994-6 (cit. on p. 41).
- [56] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann and Werner Retschitzegger. **Context-Awareness on Mobile Devices - the Hydrogen Approach**. In: *HICSS*. 2003, p. 292 (cit. on pp. 44, 52–53, 61).
- [57] Petra Hofstedt and Armin Wolf. **Einführung in die Constraint-Programmierung: Grundlagen, Methoden, Sprachen, Anwendungen**. Springer, 2007. isbn: 9783540231844. doi: 10.1007/978-3-540-68194-6 (cit. on p. 101).
- [58] Nicola Hönlé, Matthias Großmann, Daniela Nicklas and Bernhard Mitschang. **Design and implementation of a domain-aware data model for pervasive**

- context information.** In: *Computer Science - R&D* 24.1-2 (2009), pp. 69–83 (cit. on pp. 56, 61, 192).
- [59] Markus C. Huebscher and Julie A. McCann. **Adaptive middleware for context-aware applications in smart-homes.** In: *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing, Toronto, Ontario, Canada, October 18-22, 2004.* Ed. by Paddy Nixon and Fabio Kon. ACM, 2004, pp. 111–116. isbn: 1-58113-951-9 (cit. on pp. 17, 45, 49, 60, 62).
- [60] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira and Benjamin Reed. **Zoo-Keeper: wait-free coordination for internet-scale systems.** In: *Proceedings of the 2010 USENIX conference on USENIX annual technical conference.* USENIX-ATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 11–11. url: <http://dl.acm.org/citation.cfm?id=1855840.1855851> (cit. on p. 142).
- [61] IBM. **An architectural blueprint for autonomic computing.** In: *White Paper.* Ed. by IBM Corporation. 2006. url: [http://www-03.ibm.com/autonomic/pdfs/AC\\_Blueprint\\_White\\_Paper\\_4th.pdf](http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf) (cit. on pp. 24–25).
- [62] Michael C. Jaeger. **Optimising Quality-of-Service for the Composition of Electronic Services.** PhD thesis. Berlin University of Technology, Jan. 2007. url: <http://opus.kobv.de/tuberlin/volltexte/2007/1472/> (cit. on pp. 6, 135).
- [63] Jeffrey O. Kephart and Rajarshi Das. **Achieving Self-Management via Utility Functions.** In: *IEEE Internet Computing* 11.1 (2007), pp. 40–48 (cit. on p. 25).
- [64] Ralf Kernchen, David Bonnefoy, Agathe Battestini, Bernd Mrohs, Matthias Wagner and Mika Klemettinen. **Context-awareness in MobiLife.** In: *Proc. of the 15th IST Mobile Summit.* Mykonos, Greece, June 2006. url: [http://www.ist-esense.org/fileadmin/images/PDF\ \\_Other\ IST\ \\_SUMMIT\ \\_2006\ 5. c\ \\_MOBILIFE.pdf](http://www.ist-esense.org/fileadmin/images/PDF\ _Other\ IST\ _SUMMIT\ _2006\ 5. c\ _MOBILIFE.pdf) (cit. on pp. 55, 61–62).
- [65] Mohammad Ullah Khan. **Unanticipated Dynamic Adaptation of Mobile Applications.** PhD thesis. Kassel, Germany: University of Kassel, Fachbereich 16: Elektrotechnik/Informatik, Distributed Systems Group, Mar. 2010. url: <http://www.upress.uni-kassel.de/publi/abstract.php?978-3-89958-918-4> (cit. on p. 25).
- [66] Michael Kifer and Georg Lausen. **F-Logic: A Higher-Order language for Reasoning about Objects, Inheritance, and Scheme.** In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989.* Ed. by James Clifford, Bruce G. Lindsay and David Maier. ACM Press, 1989, pp. 134–146 (cit. on p. 39).
- [67] Tim Kindberg and Armando Fox. **System Software for Ubiquitous Computing.** In: *IEEE Pervasive Computing* 1 (1 Jan. 2002), pp. 70–81. issn: 1536-1268. doi: 10.1109/MPRV.2002.993146. url: <http://dl.acm.org/citation.cfm?id=612822.612834> (cit. on p. 14).
- [68] Cédric Kiss. **Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 2.0.** a WD in Last Call. <http://www.w3.org/TR/2007/WD-CCPP-struct-vocab2-20070430>. W3C, Apr. 2007 (cit. on p. 47).

- [69] Alexander Kohout. **Context Constraints in a Dynamic Context Service Discovery and Binding Process**. Bachelor Thesis. University of Kassel, June 2011 (cit. on pp. 101, 182–183).
- [70] Panu Korpiää, Jonna Häkkinä, Juha Kela, Sami Ronkainen and Ilkka Känsälä. **Utilising context ontology in mobile device application personalisation**. In: *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, MUM 2004, College Park, Maryland, USA, October 27-29, 2004*. Ed. by David S. Doermann and Ramani Duraiswami. Vol. 83. ACM International Conference Proceeding Series. ACM, 2004, pp. 133–140. isbn: 1-58113-981-0 (cit. on pp. 54, 61).
- [71] Panu Korpiää, Esko-Juhani Malm, Ilkka Salminen, Tapani Rantakokko, Vesa Kyllönen and Ilkka Känsälä. **Context management for end user development of context-aware applications**. In: *6th International Conference on Mobile Data Management (MDM 2005), Ayia Napa, Cyprus, May 9-13, 2005*. Ed. by Panos K. Chrysanthis and George Samaras. ACM, 2005, pp. 304–308. isbn: 1-59593-041-8 (cit. on pp. 54, 61).
- [72] Panu Korpiää and Jani Mäntyjärvi. **An Ontology for Mobile Device Sensor-Based Context Awareness**. In: *Modeling and Using Context, 4th International and Interdisciplinary Conference, CONTEXT 2003, Stanford, CA, USA, June 23-25, 2003, Proceedings*. Ed. by Patrick Blackburn, Chiara Ghidini, Roy M. Turner and Fausto Giunchiglia. Vol. 2680. LNCS. Springer, 2003, pp. 451–458. isbn: 3-540-40380-9 (cit. on pp. 54, 61).
- [73] P. Korpiää, J. Mäntyjärvi, J. Kela, H. Keränen and E. J. Malm. **Managing context information in mobile devices**. In: *Pervasive Computing, IEEE 2.3 (2003)*, pp. 42–51. doi: 10.1109/MPRV.2003.1228526. url: <http://dx.doi.org/10.1109/MPRV.2003.1228526> (cit. on pp. 44, 54, 61, 81).
- [74] Mustafa Kose, Ozlem Durmaz and Cem Ersoy. **Online Human Activity Recognition on Smart Phones**. In: *2nd International Workshop on Mobile Sensing Workshop co-located with IPSN '12 and CPSWEEK*. Beijing China, Apr. 2012. url: [http://research.microsoft.com/en-us/um/beijing/events/ms\\_ipsn12/papers/msipsn-kose.pdf](http://research.microsoft.com/en-us/um/beijing/events/ms_ipsn12/papers/msipsn-kose.pdf) (cit. on p. 151).
- [75] Michael Krause and Iris Hochstatter. **Challenges in Modelling and Using Quality of Context (QoC)**. In: *Mobility Aware Technologies and Applications, Second International Workshop, MATA 2005, Montreal, Canada, October 17-19, 2005, Proceedings*. Ed. by Thomas Magedanz, Ahmed Karmouch, Samuel Pierre and Iakovos S. Venieris. Vol. 3744. LNCS. Springer, 2005, pp. 324–333. isbn: 3-540-29410-4 (cit. on pp. 7, 17, 19–21, 46, 60, 62).
- [76] Jennifer R. Kwapisz, Gary M. Weiss and Samuel Moore. **Activity recognition using cell phone accelerometers**. In: *SIGKDD Explorations 12.2 (2010)*, pp. 74–82 (cit. on p. 150).
- [77] Lee W Lacy. **OWL : Representing Information Using the Web Ontology Language**. Victoria BC, Canada: Trafford Publishing, 2005. isbn: 1-4120-3448-5 (cit. on pp. 40–41).



- [78] Robert Laddaga. **Self-Adaptive Software**. In: *DARPA SOL BAA 98-12* (Dec. 1997). Original url only available in web archive [http://web.archive.org/web/19990221110757/http://www.darpa.mil/ito/Solicitations/CBD\\_9812.html](http://web.archive.org/web/19990221110757/http://www.darpa.mil/ito/Solicitations/CBD_9812.html). url: [http://www.darpa.mil/ito/Solicitations/CBD\\_9812.html](http://www.darpa.mil/ito/Solicitations/CBD_9812.html) (cit. on p. 24).
- [79] Ralph Lange, Nazario Cipriani, Lars Geiger, Matthias Großmann, Harald Weinschrott, Andreas Brodt, Matthias Wieland, Stamatia Rizou and Kurt Rothermel. **Making the World Wide Space Happen: New Challenges for the Nexus Context Platform**. In: *Seventh Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2009, 9-13 March 2009, Galveston, TX, USA*. IEEE Computer Society, 2009, pp. 1–4. isbn: 978-1-4244-3304-9 (cit. on pp. 56, 61, 192).
- [80] Marc Langheinrich and Friedemann Mattern. **Digitalisierung des Alltags. Was ist Pervasive Computing?** In: *Aus Politik und Zeitgeschichte (B 42/2003)* (13th Oct. 2003). See also Online-Version at [www.bpb.de](http://www.bpb.de), pp. 6–12 (cit. on p. 13).
- [81] Youngseol Lee and Sung-Bae Cho. **Activity Recognition Using Hierarchical Hidden Markov Models on a Smartphone with 3D Accelerometer**. In: *Hybrid Artificial Intelligent Systems - 6th International Conference, HAIS 2011, Wroclaw, Poland, May 23-25, 2011, Proceedings, Part I*. Ed. by Emilio Corchado, Marek Kurzynski and Michal Wozniak. Vol. 6678. LNCS. Springer, 2011, pp. 460–467. isbn: 978-3-642-21218-5 (cit. on p. 150).
- [82] Othmar Lehmann, Martin Bauer, Christian Becker and Daniela Nicklas. **From Home to World - Supporting Context-aware Applications through World Models**. In: *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA*. IEEE Computer Society, 2004, pp. 297–308. isbn: 0-7695-2090-1 (cit. on pp. 4–5).
- [83] Leopold Löwenheim. **Über Möglichkeiten im Relativkalkül**. In: *Mathematische Annalen* 76 (4 1915). 10.1007/BF01458217, pp. 447–470. issn: 0025-5831. url: <http://dx.doi.org/10.1007/BF01458217> (cit. on p. 101).
- [84] Kalle Lyytinen and Youngjin Yoo. **Introduction**. In: *Commun. ACM* 45.12 (2002), pp. 62–65 (cit. on p. 14).
- [85] Robert M. MacGregor. **Inside the LOOM Description Classifier**. In: *SIGART Bulletin* 2.3 (1991), pp. 88–92 (cit. on p. 39).
- [86] Atif Manzoor, Hong Linh Truong and Schahram Dustdar. **On the Evaluation of Quality of Context**. In: *Smart Sensing and Context, Third European Conference, EuroSSC 2008, Zurich, Switzerland, October 29-31, 2008. Proceedings*. Ed. by Daniel Roggen, Clemens Lombriser, Gerhard Tröster, Gerd Kortuem and Paul J. M. Havinga. Vol. 5279. LNCS. Springer, 2008, pp. 140–153. isbn: 978-3-540-88792-8 (cit. on pp. 21–23, 56, 61–62).
- [87] Atif Manzoor, Hong Linh Truong and Schahram Dustdar. **Using Quality of Context to Resolve Conflicts in Context-Aware Systems**. In: *Quality of Context*,

*First International Workshop, QuaCon 2009, Stuttgart, Germany, June 25-26, 2009. Revised Papers.* Ed. by Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger and Frank Dürr. Vol. 5786. LNCS. Springer, 2009, pp. 144–155. isbn: 978-3-642-04558-5 (cit. on pp. 56–57, 61–62).

- [88] Atif Manzoor, Hong-Linh Truong and Schahram Dustdar. **Quality of Context: Models and Applications for Context-aware Systems in Pervasive Environments.** In: *The Knowledge Engineering Review Special Issue on Web and Mobile Information Services* (2010) (cit. on pp. 20–21, 56–57, 61–62).
- [89] R. Timothy Marler and Jasbir S. Arora. **Survey of multi-objective optimization methods for engineering.** In: *Structural and Multidisciplinary Optimization* 26.6 (1st Apr. 2004), pp. 369–395. issn: 1615-147X. doi: 10.1007/s00158-003-0368-6. url: <http://dx.doi.org/10.1007/s00158-003-0368-6> (cit. on pp. 136–137).
- [90] R. Timothy Marler and Jasbir S. Arora. **The weighted sum method for multi-objective optimization: new insights.** In: *Structural and Multidisciplinary Optimization* 41 (6 2010). 10.1007/s00158-009-0460-7, pp. 853–862. issn: 1615-147X. url: <http://dx.doi.org/10.1007/s00158-009-0460-7> (cit. on pp. 136–137).
- [91] Fabio Massacci. **The proof complexity of analytic and clausal tableaux.** In: *Theor. Comput. Sci.* 243.1-2 (2000), pp. 477–487 (cit. on p. 110).
- [92] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten and Betty H. C. Cheng. **Composing Adaptive Software.** In: *IEEE Computer* 37.7 (2004), pp. 56–64 (cit. on p. 24).
- [93] Anton Michlmayr, Florian Rosenberg, Philipp Leitner and Schahram Dustdar. **End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo.** In: *IEEE T. Services Computing* 3.3 (2010), pp. 193–205 (cit. on p. 114).
- [94] Kaisa Miettinen. **Nonlinear Multiobjective Optimization.** Vol. 12. International Series in Operations Research and Management Science. Kluwer Academic Publishers, Dordrecht, 1999 (cit. on pp. 113, 122, 136).
- [95] Enrico Motta. **Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving.** 1st. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1999. isbn: 1586030035 (cit. on p. 39).
- [96] **MUSIC – Mobile Users in Ubiquitous Computing Environments.** url: <http://ist-music.berlios.de> (cit. on pp. 6, 55).
- [97] Daniela Nicklas and Bernhard Mitschang. **On building location aware applications using an open platform based on the NEXUS Augmented World Model.** In: *Software and System Modeling* 3.4 (2004), pp. 303–313 (cit. on pp. 56, 61, 192).
- [98] Eila Niemelä and Juhani Latvakoski. **Survey of requirements and solutions for ubiquitous software.** In: *Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, MUM 2004, College Park, Maryland, USA,*

October 27-29, 2004. Ed. by David S. Doermann and Ramani Duraiswami. Vol. 83. ACM International Conference Proceeding Series. ACM, 2004, pp. 71–78. isbn: 1-58113-981-0 (cit. on pp. 14–15).

- [99] Russel Nzekwa, Romain Rouvoy and Lionel Seinturier. **Modelling Feedback Control Loops for Self-Adaptive Systems**. In: *ECEASST 28* (2010) (cit. on p. 24).
- [100] Nicole Oldham, Kunal Verma, Amit P. Sheth and Farshad Hakimpour. **Semantic WS-agreement partner selection**. In: *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*. Ed. by Les Carr, David De Roure, Arun Iyengar, Carole A. Goble and Michael Dahlin. ACM, 2006, pp. 697–706. isbn: 1-59593-323-9 (cit. on p. 92).
- [101] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum and Alexander L. Wolf. **An Architecture-Based Approach to Self-Adaptive Software**. In: *IEEE Intelligent Systems* 14.3 (May 1999), pp. 54–62. issn: 1541-1672. doi: 10.1109/5254.769885. url: <http://dx.doi.org/10.1109/5254.769885> (cit. on p. 24).
- [102] OSGi Alliance. **OSGi Service Platform Release 4**. 2007. url: <http://www.osgi.org/Main/HomePage> (cit. on pp. 55, 77, 143).
- [103] Gabor Paller. **Motion Recognition with Android Devices**. Talk/Slides at Droidcon London. Oct. 2011. url: <http://www.slideshare.net/paller/motion-recognition-with-android-devices> (cit. on pp. 149, 173).
- [104] Nearchos Paspallis. **Middleware-based development of context-aware applications with reusable components**. PhD thesis. Nicosia, Cyprus: Department of Computer Science, University of Cyprus, 2009. url: <http://www.cs.ucy.ac.cy/~paspalli/phd/index.html> (cit. on pp. 6, 55, 61–62, 143, 191).
- [105] **Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA**. IEEE Computer Society, 2004. isbn: 0-7695-2090-1.
- [106] Anand Ranganathan and Roy H. Campbell. **A Middleware for Context-Aware Agents in Ubiquitous Computing Environments**. In: *Middleware 2003, ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 16-20, 2003, Proceedings*. Ed. by Markus Endler and Douglas C. Schmidt. Vol. 2672. LNCS. Springer, 2003, pp. 143–161. isbn: 3-540-40317-5 (cit. on pp. 52, 60).
- [107] Anand Ranganathan, Jalal Al-Muhtadi and Roy H. Campbell. **Reasoning about Uncertain Contexts in Pervasive Computing Environments**. In: *IEEE Pervasive Computing* 3.2 (2004), pp. 62–70 (cit. on pp. 52, 60).
- [108] Mohammad Abdur Razzaque, Simon Dobson and Paddy Nixon. **Categorization and Modelling of Quality in Context Information**. In: *Proceedings of the IJCAI 2005 Workshop on AI and Autonomic Communications*. 2005 (cit. on p. 17).

- [109] Roland Reichle. **Information Exchange and Fusion in Dynamic and Heterogeneous Distributed Environments**. PhD thesis. Kassel, Germany: University of Kassel, Fachbereich 16: Elektrotechnik/Informatik, Distributed Systems Group, July 2010. url: <http://nbn-resolving.de/urn:nbn:de:hebis:34-2010121035166> (cit. on pp. 6, 53, 61–62, 69, 74, 80, 83, 88, 90, 191–192).
- [110] Manuel Roman, Christopher Hess, Renato Cerqueira, Roy H. Campbell and Klara Nahrstedt. **Gaia: A Middleware Infrastructure to Enable Active Spaces**. In: *IEEE Pervasive Computing* 1 (2002), pp. 74–83 (cit. on pp. 44, 52, 60).
- [111] Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger and Frank Dürr, eds. **Quality of Context, First International Workshop, QuaCon 2009, Stuttgart, Germany, June 25-26, 2009. Revised Papers**. Vol. 5786. LNCS. Springer, 2009. isbn: 978-3-642-04558-5.
- [112] Kurt Rothermel et al. **SFB 627 - Umgebungsmodelle für mobile kontextbezogene Systeme**. In: *Inform., Forsch. Entwickl.* 21.1-2 (2006), pp. 105–113 (cit. on pp. 56, 61, 192).
- [113] Romain Rouvoy, Denis Conan and Lionel Seinturier. **Software Architecture Patterns for a Context-Processing Middleware Framework**. In: *IEEE Distributed Systems Online* 9.6 (2008) (cit. on pp. 50–51, 60, 150).
- [114] Nirmalya Roy, Sajal K. Das and Christine Julien. **Resource-Optimized Quality-Assured Ambiguous Context Mediation Framework in Pervasive Environments**. In: *IEEE Trans. Mob. Comput.* 11.2 (2012), pp. 218–229 (cit. on pp. 58–59, 61–62).
- [115] Nirmalya Roy, Tao Gu and Sajal K. Das. **Supporting pervasive computing applications with active context fusion and semantic context delivery**. In: *Pervasive and Mobile Computing* 6.1 (2010), pp. 21–42 (cit. on pp. 58–59, 61–62).
- [116] Debashis Saha and Amitava Mukherjee. **Pervasive Computing: A Paradigm for the 21st Century**. In: *IEEE Computer* 36.3 (2003), pp. 25–31 (cit. on pp. 14–15).
- [117] Daniel Salber, Anind K. Dey and Gregory D. Abowd. **The Context Toolkit: Aiding the Development of Context-Enabled Applications**. In: *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15-20, 1999*. Ed. by Marian G. Williams and Mark W. Altom. ACM, 1999, pp. 434–441. isbn: 0-201-48559-1 (cit. on pp. 44, 49, 60).
- [118] Mazeiar Salehie and Ladan Tahvildari. **Self-adaptive software: Landscape and research challenges**. In: *TAAS* 4.2 (2009) (cit. on p. 25).
- [119] M. Satyanarayanan. **Pervasive Computing: Vision and Challenges**. In: *IEEE Personal Communications* 8 (2001), pp. 10–17 (cit. on pp. 14–15).
- [120] Gregor Schiele, Marcus Handte and Christian Becker. **Pervasive Computing Middleware**. In: *Handbook of Ambient Intelligence and Smart Environments*. Ed. by Hideyuki Nakashima, Hamid Aghajan and Juan Carlos Augusto. Springer US, 2010, pp. 201–227. isbn: 978-0-387-93808-0. url: [http://dx.doi.org/10.1007/978-0-387-93808-0\\_8](http://dx.doi.org/10.1007/978-0-387-93808-0_8) (cit. on p. 4).

- [121] Bill Schilit, Norman Adams and Roy Want. **Context-Aware Computing Applications**. In: *Proceedings of the Workshop on Mobile Computing Systems and Applications*. IEEE Computer Society, 1994, pp. 85–90 (cit. on p. 16).
- [122] Uwe Schöning. **Logik für Informatiker (4. Aufl.)** Reihe Informatik. Spektrum Akademischer Verlag, 1995, pp. 1–207. isbn: 978-3-86025-684-8 (cit. on pp. 27–28, 32–35).
- [123] Kamran Sheikh, Maarten Wegdam and Marten van Sinderen. **Middleware Support for Quality of Context in Pervasive Context-Aware Systems**. In: *Fifth Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2007), 19-23 March 2007, White Plains, New York, USA*. IEEE Computer Society, 2007, pp. 461–466. isbn: 978-0-7695-2788-8 (cit. on pp. 20–23, 46, 60).
- [124] Kamran Sheikh, Maarten Wegdam and Marten van Sinderen. **Quality-of-Context and its use for Protecting Privacy in Context Aware Systems**. In: *JSW 3.3 (2008)*, pp. 83–93 (cit. on pp. 46, 60).
- [125] M.J. van Sinderen, A.T. van Halteren, M. Wegdam, H.B. Meeuwissen and E.H. Eertink. **Supporting context-aware mobile applications: an infrastructure approach**. In: *Communications Magazine, IEEE* 44.9 (Sept. 2006), pp. 96–104. issn: 0163-6804. doi: 10.1109/MCOM.2006.1705985 (cit. on pp. 46, 60).
- [126] N. J. A. Sloane. **The On-Line Encyclopedia of Integer Sequences – A000522**. <http://oeis.org/A000522>. Oct. 2007 (cit. on p. 177).
- [127] N. J. A. Sloane, Robert G. Wilson and Rick L. Shepherd. **The On-Line Encyclopedia of Integer Sequences – A007526**. <http://oeis.org/A007526>. June 2005 (cit. on p. 178).
- [128] Raymond Smullyan. **First-Order Logic**. Mineola: Dover, 1995. isbn: 978-0486683706 (cit. on pp. 28–29, 32, 36, 110).
- [129] Ahmet Soyly, Patrick De Causmaecker and Piet Desmet. **Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering**. In: *JSW 4.9 (2009)*, pp. 992–1013 (cit. on p. 14).
- [130] Steffen Staab and Rudi Studer, eds. **Handbook on Ontologies**. Second Edition. International Handbooks on Information Systems. Springer, 2009. isbn: 978-3-540-70999-2 (cit. on pp. 38, 41, 93).
- [131] Dave Steinberg, Frank Budinsky, Marcelo Patenostro and Ed Merks. **EMF: Eclipse Modeling Framework, 2nd Edition**. Addison Wesley, 2008 (cit. on p. 88).
- [132] Ralf Stephan. **The On-Line Encyclopedia of Integer Sequences – A093964**. <http://oeis.org/A093964>. Apr. 2004 (cit. on p. 178).
- [133] Thomas Strang and Claudia Linnhoff-Popien. **A Context Modeling Survey**. In: *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*. 2004. url: <http://elib.dlr.de/7444/1/UbiComp2004ContextWSCameraReadyVersion.pdf> (cit. on pp. 44, 81).

- [134] Thomas Strang, Claudia Linnhoff-Popien and Korbinian Frank. **CoOL: A Context Ontology Language to Enable Contextual Interoperability**. In: *Distributed Applications and Interoperable Systems, 4th IFIP WG6.1 International Conference, DAIS 2003, Paris, France, November 17-21, 2003, Proceedings*. Ed. by Jean-Bernard Stefani, Isabelle M. Demeure and Daniel Hagimont. Vol. 2893. LNCS. Springer, 2003, pp. 236–247. isbn: 3-540-20529-2 (cit. on pp. 8, 45–46, 60, 62, 71, 77, 81, 190–191).
- [135] **Semantic Web for Earth and Environmental Terminology (SWEET)**. <http://sweet.jpl.nasa.gov/index.html> (accessed 2011-09-08). NASA (cit. on p. 73).
- [136] Graham Upton and Ian Cook. **Understanding statistics**. Oxford University Press, 1996. isbn: 9780199143917. url: [http://books.google.com/books?id=vXzWG09\\\_SzAC](http://books.google.com/books?id=vXzWG09\_SzAC) (cit. on p. 125).
- [137] Claudia Villalonga, Daniel Roggen, Clemens Lombriser, Piero Zappi and Gerhard Tröster. **Bringing Quality of Context into Wearable Human Activity Recognition Systems**. In: *Quality of Context, First International Workshop, QuaCon 2009, Stuttgart, Germany, June 25-26, 2009. Revised Papers*. Ed. by Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger and Frank Dürr. Vol. 5786. LNCS. Springer, 2009, pp. 164–173. isbn: 978-3-642-04558-5 (cit. on p. 23).
- [138] W3C. **Semantic Web Activity Page**. Ed. by W3C. 2011. url: <http://www.w3.org/2001/sw/> (cit. on p. 39).
- [139] OWL Working Group W3C. **OWL 2 Web Ontology Language: Document Overview**. W3C Recommendation, Oct. 2009. url: <http://www.w3.org/TR/owl2-overview/> (cit. on pp. 39, 41, 70).
- [140] OWL Working Group W3C. **OWL 2 Web Ontology Language: Primer**. Ed. by Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider and Sebastian Rudolph. W3C Recommendation, Oct. 2009. url: <http://www.w3.org/TR/owl2-primer/> (cit. on pp. 39, 41, 70).
- [141] RDF Working Group W3C. **RDF Primer**. Ed. by Frank Manola and Eric Miller. W3C Recommendation, Feb. 2004. url: <http://www.w3.org/TR/rdf-primer/> (cit. on pp. 39–40).
- [142] RDF Working Group W3C. **RDF Vocabulary Description Language 1.0: RDF Schema**. Ed. by Dan Brickley and R.V. Guha. W3C Recommendation, Feb. 2004. url: <http://www.w3.org/TR/rdf-schema/> (cit. on p. 39).
- [143] WSDL Working Group W3C. **Web Service Definition Language (WSDL)**. Ed. by Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana. W3C Recommendation, Mar. 2001. url: <http://www.w3.org/TR/wsdl> (cit. on p. 90).
- [144] Yi Wang, Jialiu Lin, Murali Annavaram, Quinn Jacobson, Jason I. Hong, Bhaskar Krishnamachari and Norman M. Sadeh. **A framework of energy efficient mobile sensing for automatic user state recognition**. In: *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys 2009), Kraków, Poland, June 22-25, 2009*. Ed. by Krzysztof Zielinski, Adam

- Wolisz, Jason Flinn and Anthony LaMarca. ACM, 2009, pp. 179–192. isbn: 978-1-60558-566-6 (cit. on pp. 5, 51, 60, 62, 151).
- [145] Mark Weiser. **The Computer for the 21st Century**. In: *Scientific American Communications, Computers, and Network* (Sept. 1991). url: <http://www.ics.uci.edu/~dutt/ics212-wq05/weiser-sci-am-sep-91.pdf> (cit. on pp. 3, 13–15).
- [146] Mark Weiser and John Seely Brown. **Designing Calm Technology**. In: *PowerGrid Journal* (July 1996). url: <http://www.johnseelybrown.com/calmtech> (cit. on p. 15).
- [147] Thomas Weise. **Global Optimization Algorithms – Theory and Application**. it-weise.de (self-published): Germany, 2009. url: <http://www.it-weise.de/projects/book.pdf> (cit. on p. 137).
- [148] Kun Yang, Alex Galis and Hsiao-Hwa Chen. **QoS-Aware Service Selection Algorithms for Pervasive Service Composition in Mobile Wireless Environments**. In: *MONET* 15.4 (2010), pp. 488–501 (cit. on pp. 6, 135).
- [149] Zhixian Yan, Vigneshwaran Subbaraju, Dipanjan Chakraborty, Archan Misra and Karl Aberer. **Energy-Efficient Continuous Activity Recognition on Mobile Phones: An Activity-Adaptive Approach**. In: *16th IEEE International Symposium on Wearable Computers (ISWC 2012), 18-22 June 2012, Newcastle, UK, 2012* (cit. on pp. 151–152).

## C.2 Publications as (Co-)Author

- [150] Alisa Devlic, Roland Reichle, Michael Wagner, Manuele Kirsch Pinheiro, Yves Vanrompay, Yolande Berbers and Massimo Valla. **Context inference of users' social relationships and distributed policy management**. In: *6th IEEE Workshop on Context Modeling and Reasoning (CoMoRea) at the 7th IEEE International Conference on Pervasive Computing and Communication (PerCom'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8.
- [151] Jacqueline Floch, Cristina Fra, Rolf Fricke, Kurt Geihs, Michael Wagner, Jorge Lorenzo, Eduardo Soladana, Stefan Mehlhase, Nearchos Paspallis, Hossein Rahnama, Pedro A. Ruiz and Ulrich Scholz. **Playing MUSIC – building context-aware and self-adaptive mobile applications**. In: *Software: Practice and Experience* (2012). issn: 1097-024X. doi: 10.1002/spe.2116. url: <http://dx.doi.org/10.1002/spe.2116> (cit. on pp. 6, 55).
- [152] Kurt Geihs, Christoph Evers, Roland Reichle, Michael Wagner and Mohammad Ullah Khan. **Development support for QoS-aware service-adaptation in ubiquitous computing applications**. In: *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*. Ed. by William C. Chu, W. Eric Wong, Mathew J. Palakal and Chih-Cheng Hung. ACM, 2011, pp. 197–202. isbn: 978-1-4503-0113-8 (cit. on p. 10).
- [153] Kurt Geihs, Roland Reichle, Michael Wagner and Mohammad Ullah Khan. **Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and**

- Service-Oriented Environments.** In: Software Engineering for Self-Adaptive Systems. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 146–163. url: <http://www.vs.uni-kassel.de/publications/2009/GRWK09>.
- [154] Kurt Geihs, Roland Reichle, Michael Wagner and Mohammad Ullah Khan. **Service-Oriented Adaptation in Ubiquitous Computing Environments.** In: *Proceedings IEEE CSE'09, 12th IEEE International Conference on Computational Science and Engineering, August 29-31, 2009, Vancouver, BC, Canada.* IEEE Computer Society, 2009, pp. 458–463 (cit. on pp. 10, 25).
- [155] Gabriel Hermosillo, Russel Nzekwa and Michael Wagner, eds. **Context-Aware Adaptation Mechanism for Pervasive and Ubiquitous Services 2011.** Vol. 43. Electronic Communications of EASST (ECEASST), June 2011. url: <http://journal.ub.tu-berlin.de/eceasst/article/view/584/615>.
- [156] Mohammad U. Khan, Roland Reichle, Michael Wagner, Kurt Geihs, Ulrich Scholz, Constantinos Kakousis and George A. Papadopoulos. **An Adaptation Reasoning Approach for Large Scale Component-based Applications.** In: *Communications of the EASST.* Vol. 19. Proceedings of the Second International DisCoTec Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services(CAMPUS 2009). 2009.
- [157] Sonia Ben Mokhtar, Romain Rouvoy and Michael Wagner, eds. **Context-Aware Adaptation Mechanism for Pervasive and Ubiquitous Services 2010.** Vol. 28. Electronic Communications of EASST (ECEASST), June 2010. url: <http://journal.ub.tu-berlin.de/index.php/eceasst/issue/view/38>.
- [158] Roland Reichle, Michael Wagner, Mohammad Ullah Khan, Kurt Geihs, Jorge Lorenzo, Massimo Valla, Cristina Fra, Nearchos Paspallis and George A. Papadopoulos. **A Comprehensive Context Modeling Framework for Pervasive Computing Systems.** In: *Distributed Applications and Interoperable Systems, 8th IFIP WG 6.1 International Conference, DAIS 2008, Oslo, Norway, June 4-6, 2008. Proceedings.* Ed. by René Meier and Sotirios Terzis. Vol. 5053. LNCS. 2008, pp. 281–295. isbn: 978-3-540-68639-2 (cit. on pp. 10, 53, 55, 69, 74, 76, 81, 191).
- [159] Roland Reichle, Michael Wagner, Mohammad Ullah Khan, Kurt Geihs, Massimo Valla, Cristina Fra, Nearchos Paspallis and George A. Papadopoulos. **A Context Query Language for Pervasive Computing Environments.** In: *Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom 2008), 17-21 March 2008, Hong Kong.* IEEE Computer Society, 2008, pp. 434–440 (cit. on pp. 10, 53, 55–56, 83, 88, 90).
- [160] Romain Rouvoy, Mauro Caporuscio and Michael Wagner, eds. **Context-aware Adaption Mechanisms for Pervasive and Ubiquitous Services.** Vol. 11. Electronic Communications of EASST (ECEASST), June 2008. url: <http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/18>.
- [161] Romain Rouvoy and Michael Wagner, eds. **Context-Aware Adaptation Mechanism for Pervasive and Ubiquitous Services 2009.** Vol. 19. Electronic Communications of EASST (ECEASST), June 2009. url: <http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/25>.



- [162] Hendrik Skubch, Michael Wagner and Roland Reichle. **A Language for Interactive Cooperative Agents**. Tech. rep. University of Kassel, 2009. url: <http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2009032026745>.
- [163] Hendrik Skubch, Michael Wagner, Roland Reichle, Stefan Triller and Kurt Geihs. **Towards a Comprehensive Teamwork Model for Highly Dynamic Domains**. In: *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence*. 2010.
- [164] Martin Steinebach, Michael Wagner and Patrick Wolf. **Verteilte Suche nach digitalen Wasserzeichen in eMule**. In: *D-A-CH Security 2007*. Klagenfurt: Patrick Horster, June 2007, pp. 519–530.
- [165] Michael Wagner. **Context as a service**. In: *UbiComp 2010: Ubiquitous Computing, 12th International Conference, UbiComp 2010, Copenhagen, Denmark, September 26-29, 2010, Adjunct Papers Proceedings*. Ed. by Jakob E. Bardram, Marc Langheinrich, Khai N. Truong and Paddy Nixon. ACM International Conference Proceeding Series. ACM, 2010, pp. 489–492. isbn: 978-1-4503-0283-8 (cit. on p. 10).
- [166] Michael Wagner. **Verteilte Suche nach digitalen Wasserzeichen in eMule**. In: *Informatik-Spektrum* 30.4 (Aug. 2007), pp. 264–272. url: <http://dx.doi.org/10.1007/s00287-007-0162-8>.
- [167] Michael Wagner, Dieter Hogrefe, Kurt Geihs and Klaus David, eds. **Workshops der Wissenschaftlichen Konferenz Kommunikation in verteilten Systemen 2009 in Kassel (WowKiVS 2009)**. Vol. 17. EASST, 2009. url: <http://eecasst.cs.tu-berlin.de/index.php/eecasst/issue/view/24>.
- [168] Michael Wagner, Roland Reichle and Kurt Geihs. **Context as a service - Requirements, design and middleware support**. In: *Ninth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2011, 21-25 March 2011, Seattle, WA, USA, Workshop Proceedings*. IEEE, 2011, pp. 220–225 (cit. on p. 10).