An Adaptive, Searchable and Extendable Context Model, enabling cross-domain Context Storage, Retrieval and Reasoning
Architecture, Design, Implementation and Discussion

Felix Dobslaw
Acknowledgement

I would like to thank my first academic advisor Theo Kanter for inspiring discussions and my second academic advisor Åke Malmberg for all the efforts, arranging a very pleasant room for me in Östersund and integrating me into the local staff. Special thanks to Annika Berggren and Patrik Österberg, who helped me a lot with the administrative work. Thanks also to Aron Larsson for the very diverse discussions and brainstorming sessions.

Furthermore I would like to thank the ITM department at the MidSweden University for covering my travel expenses.

All your help is well appreciated.
## Table of Contents

1 Introduction ........................ 1
   1.1 Importance of Context .................. 2
       1.1.1 Theory and Practice .................. 3
   1.2 Context of this Thesis .................. 4
   1.3 Problem Statement ...................... 5
   1.4 Related Work ........................ 6
   1.5 Context on the Internet ................. 9
   1.6 Thesis Structure ....................... 9

2 Context Modeling ..................... 11
   2.1 Ontologies .......................... 11
       2.1.1 Inter-Lingua ....................... 12
   2.2 The Approaches ......................... 12
   2.3 The CII(Context Information Integration)-model ............ 14
       2.3.1 How Context is expressed .......... 16
       2.3.2 Subconcepts and Inheritance .... 18

3 Architecture and Implementation ...... 20
   3.1 General Architecture .................. 20
       3.1.1 The Context Controller .......... 22
       3.1.2 Service Enabling .................. 22
   3.2 Application Programming Interfaces ...... 23
       3.2.1 Context Provider .................. 24
       3.2.2 Context Consumer ................. 26
   3.3 Implementation ....................... 26
       3.3.1 The DataNucleus Access Platform .. 27
       3.3.2 Proof of Concept .................. 30
       3.4 The Context Controller .............. 31

4 Context Retrieval ..................... 33
   4.1 Language ........................... 33
       4.1.1 Existing Approaches ............... 33
       4.1.2 JDOQL .......................... 34
   4.2 Nearest Neighbors ..................... 35
       4.2.1 Searching In Ranges ................ 36

5 Discussion .......................... 41
   5.1 CII-Model ........................... 41
       5.1.1 General .......................... 41
       5.1.2 Flexibility ....................... 42
   5.2 Approach ............................ 43
       5.2.1 General ......................... 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.2 Extendability</td>
<td>46</td>
</tr>
<tr>
<td>5.2.3 Openness</td>
<td>46</td>
</tr>
<tr>
<td>5.2.4 Reasoning</td>
<td>46</td>
</tr>
<tr>
<td>5.2.5 Security</td>
<td>47</td>
</tr>
<tr>
<td>5.2.6 Scalability</td>
<td>47</td>
</tr>
<tr>
<td>5.2.7 Performance</td>
<td>47</td>
</tr>
<tr>
<td>5.2.8 Flexibility</td>
<td>49</td>
</tr>
<tr>
<td>5.2.9 Searchability</td>
<td>49</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>52</td>
</tr>
<tr>
<td>6.1 Future Work</td>
<td>53</td>
</tr>
<tr>
<td>6.1.1 CII on top of DCXP</td>
<td>53</td>
</tr>
<tr>
<td>6.1.2 Implementation</td>
<td>54</td>
</tr>
<tr>
<td>References</td>
<td>56</td>
</tr>
<tr>
<td>A Use Cases</td>
<td>59</td>
</tr>
<tr>
<td>A.1 Traffic Information System</td>
<td>59</td>
</tr>
<tr>
<td>A.1.1 Traffic Service</td>
<td>60</td>
</tr>
<tr>
<td>A.1.2 Domain Model</td>
<td>61</td>
</tr>
<tr>
<td>A.2 Ad-Hoc History Retrieval</td>
<td>70</td>
</tr>
<tr>
<td>A.2.1 Ad-hoc Service</td>
<td>71</td>
</tr>
<tr>
<td>A.2.2 Domain Model</td>
<td>71</td>
</tr>
<tr>
<td>A.3 Messenger Brokering</td>
<td>72</td>
</tr>
<tr>
<td>A.3.1 Brokering Service</td>
<td>72</td>
</tr>
<tr>
<td>A.3.2 Domain Model</td>
<td>74</td>
</tr>
<tr>
<td>B Tutorial</td>
<td>75</td>
</tr>
<tr>
<td>B.1 Step by Step Guide</td>
<td>75</td>
</tr>
<tr>
<td>B.1.1 Creating the Database</td>
<td>75</td>
</tr>
<tr>
<td>B.1.2 Creating the Application</td>
<td>76</td>
</tr>
<tr>
<td>B.2 What is What</td>
<td>77</td>
</tr>
<tr>
<td>B.2.1 lib-folder (1)</td>
<td>78</td>
</tr>
<tr>
<td>B.2.2 lib-folder (2)</td>
<td>78</td>
</tr>
<tr>
<td>B.2.3 files</td>
<td>78</td>
</tr>
<tr>
<td>C Software Design and Implementation</td>
<td>80</td>
</tr>
<tr>
<td>C.1 Components</td>
<td>80</td>
</tr>
<tr>
<td>C.1.1 Model</td>
<td>80</td>
</tr>
<tr>
<td>C.1.2 API</td>
<td>81</td>
</tr>
<tr>
<td>C.1.3 Persistence</td>
<td>81</td>
</tr>
<tr>
<td>C.2 Domain Context Management</td>
<td>87</td>
</tr>
</tbody>
</table>
Abbreviations

2PC: Two Phase Commit Protocol
3GPP: Third Generation Partnership Project
ACID: Atomicity, Consistency, Isolation, Durability
ACL: Access Control Lists
API: Application Programming Interface
ASC: Aspect Scale Context (Model)
CII: Context Information Integration (Model)
CoBrA: Context Broker Architecture
CXP: Context Exchange Protocol
db4o: Database For Objects
DCXP: Distributed Context Exchange Protocol
DTD: Document Type Definition
E/R: Entity Relationship (Model)
FLORID: F-LOgic Reasoning In Databases
HTTP: Hypertext Transfer Protocol
IMS: IP Multimedia System
GGSN: Gateway GPRS Support Node
GSM: Global System for Mobile Communication
GSN: GGSN + SGSN
GPRS: General Packet Radio Services
GPS: Global Positioning System
GUI: Graphical User Interface
ICQ: I Seek You
JCAF: Java Context Framework
JDBC: Java Database Connectivity
JDK: Java Development Kit
JDO: Java Database Objects
JDOQL: Java Data Objects Query Language
JPA: Java Persistence API
JPQL: Java Persistence Querying Language
JVM: Java Virtual Machine
Abbreviations

LSH: Locality Sensitive Hashing
MSP: Media Sessions Protocol
NP: Nondeterministic Polynomial
OBMG: Object Management Group
OODBMS: Object Oriented Database Management System
OLAP: Online Analytical Processing
OLTP: Online Transactional Processing
OQL: Object Querying Language
OS: Operating System
OSGi: Open Services Gateway Initiative
OWL: Web Ontology Language
REST: Representational State Transfer
RDBMS: Relational Database Management System
RDF: Resource Description Framework
RDFS: RDF-Schema
RMI: Remote Method Invocation
SBG: Session Border Gateway
SGSN: Serving Gateway Support Node
SIP: Session Initialization Protocol
SIMPLE: Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions
SOAP: Simple Object Access Protocol
SPARQL: SPARQL Protocol and RDF Query Language
SQL: Structured Querying Language
SVN: Subversion
UCI: Unified Context Identifier
UDP: User Datagram Protocol
UML: Unified Modeling Language
URI: Unified Resource Identifier
W3C: World Wide Web Consortium
WSGN: Wireless Sensor Network Gateway
XML: Extensible Markup Language
Abstract

The specification of communication standards and increased availability of sensors to mobile phones and mobile systems are responsible for a significantly growing sensor availability in populated environments. These devices are able to measure physical parameters and make this data available via communication in sensor networks. To take advantage of the so acquiring information for public services, other parties have to be able to receive and interpret it. Locally measured data could be seen as a means to describe user context. For a generic processing of arbitrary context data, a model for the specification of environments, users, information sources and information semantics has to be defined. Such a model would in the optimal case enable global domain crossing context usage and hence a broader foundation for context interpretation and integration. This thesis proposes the CII-(Context Information Integration) model for the persistence and retrieval of context information in mobile, dynamically changing, environments. It discusses the terms context and context modeling under the analysis of former publications in the field. Furthermore an architecture and prototype are presented. Live and historical data are stored and accessed by the same platform and querying processor, but under the hood treated in an optimized fashion. Optimized retrieval for closeness in n-dimensional context-spaces is supported by a dedicated method. The implementation enables self-aware, shareable agents that are able to reason or act based upon the global context, including their own. These agents can be considered a part of the whole context, being movable and executable for all context-aware applications. By applying open source technology, a gratifying implementation of CII is feasible. The document contains a thorough discussion about the software design and further prototype development. The use cases in the end of the document show the flexibility and extendability of the model and its implementation as a context-base for three entirely different applications.
1 Introduction

This thesis communicates a new general approach to store and process context data. The proposed mechanism is highly influenced by the requirements of ubiquitous computing and context-awareness. The field of ubiquitous computing is also known as pervasive computing or ambient intelligence. It is a model for human computer interaction that embeds technology into everyday life in a way that active and direct user-input is reduced to a minimum. The concept of context-awareness refers to the idea that computers can both sense and react based on their environment.

For ubiquitous computing, these environments should provide a means to maintain, share, protect, query and discover data. Mobile devices with their sensing and transmitting units generally don’t remain in a static position, by far don’t have 100% up-time and do have significant boundaries in computational power, as well as life-time.

The main part of this thesis deals with how context is maintained, shared and queried. It abstracts from security issues like authentication and authorization, which nevertheless are of great importance to ubiquitous computing.

For now, the focus lies in the context(-data) itself. Therefore a definition of what context is will be given. The definition, universally accepted by computer scientists:

Definition 1. Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. [1]

Kohvakko argues that this definition only holds for certain domains, ”that the authors are talking about entity as an object” where ”it is clear that an event or a situation...among other concepts, can also have context,...” [2]. For the purpose of this thesis, Dey’s definition matches. In object oriented design, which is applied, everything can be considered an object. This includes for instance events or situations. Anyway, since interaction not only remains between user and application, but between applications, the definitions last sentence will be modified to

...the interaction between a user and an application or between applications, including the user and applications themselves.

For context to be applied in a context-aware environment, it requires a certain representation. Since context depends on the eye of the beholder, an explicit definition that suits all prospects can hardly be given. However, for
the propagation and generic interpretation, global context requires a common model that maintains a general structure as a means for reuseability, interoperability and retrievability. Context modeling deals with the questions and problems regarding these challenges.

That is the reason why the terms context and context modeling are discussed in this thesis, before concrete questions regarding the chosen context modeling approach are discussed.

1.1 Importance of Context

The term context was introduced in computer science when Schili et al in [3] discussed context-awareness in computing applications. Devices, for instance mobile phones, could be context-aware. They sense their environment by build in sensors and react by triggering actions or forwarding the context data via standard protocols to other devices. Within a globally accessible data-structure, this context can serve as a driver for innovative distributed context-aware applications that ground on the derivation of user interests or needs. Examples might be the tracking of GPS-sensor (Global Positioning System) equipped humans, spilled by an avalanche or a lunch-service that recommends nearby restaurants, comprising daytime and current location for a traveling salesman.

That the term context was brought into discussion in 1994 does not mean that it was not used as a principle before. In the beginning context was mainly seen as information with respect to the location of entities. Over the years a consensus has been reached that context is more than that. Bolchini et al enumerate in [4] the five key drivers for context-aware computing, namely:

1. **adapt interfaces**: modify user interfaces for optimized usage.
2. **tailor the set of application relevant data**: reduce the amount of data to what is really relevant.
3. **discover services**, possibly of interest to the user.
4. **make the user interaction implicit**: tracking it’s behavior and deriving possible future activities.
5. **build smart environments**: dynamically changing environments
6. to improve the users performance or happiness.

These items don’t only address locational data, but take into account user purposes, desires and needs.

Context-aware computing was starting out from within the human-computer interaction community. It is regarded one of the key functionalities in ubiquitous computing. Obviously, the (intelligent) anticipation of user desires by an application could reduce the need for active user interaction. Having access to and a means to understand user context is considered a required basis to reach this goal.
1.1.1 Theory and Practice

Today there are many applications around that are (claiming to be) context-aware. Many of these use domain specific formats to represent context ([5], [6], [7]). In general, protocols for the context exchange are tightly coupled to the domain and its terms. Further more they propose/assume a specific system architecture including persistence and service consumption. Of course, these steps are taken to optimize applicability and performance for the respective services. One disadvantage in this is that context information from dissimilar applications or sources is not integrable, and can therefore not be used as a foundation for cross-domain services or reasoning.

Consequentially researchers were after a solution for a general representation of context ([2], [8], [9]), so to make it integrable. The semantic web movement can in some sense be included into the approaches. It was thought to be the semantic extension of the Internet, improving the understanding and satisfaction of users and their requests, while enabling the capability of automated reasoning upon commonly followed structures. The semantic web ontology representation standard OWL (Web Ontology Language) in combination with RDF (Resource Description Framework) were found to be suiting for the organization of highly decentralized information on the Internet.

In return, the semantic web achievements could be used in the area of ubiquitous computing. An integrability with the Internet standards would be guaranteed. Frameworks like Protege [10] were developed for the modeling of meta-models; so called ontologies (discussed in chapter two). Their integration into existing programming languages (e.g. [11] for Java) and their persistence (e.g. Sesame [12]), were approached. It exists a large amount of tools, supporting modeling with respect to semantics, using the semantic web standard. One of the biggest problems remaining is the scalability and performance of this highly dynamic models. More discussion about this matter will follow.

However, a universally accepted general purpose context model is not present yet. Many approaches head into specialized directions (generality, live-time constraints for searches, languages) but the golden rule has not yet been found. Kohvakko states that ”the future of ambient and context networks lies in efficient storage, management & distributed context aware processing”[2]. He states as well that the approach ”from application to theory doesn’t work here” [2]. The implementation of diverging context middlewares does not help solving the problem of integrability.

Therefore this research enforces the finding of a general model for the specification of context, a suiting system architecture and related prototyping. Preferably the model should be able to integrate existing models and especially being applicable with respect to sensor networks. As a flexible framework, it should be independent from the middleware for context distribution. The treatment of live data as well as historical data should be included. Where the live data storage requires optimization for real-time usage, the historical data requires data structures and mechanisms for data mining and reasoning.
1 Introduction

Figure 1.2.1: The MediaSense infrastructure which is about to be built. This thesis discusses the blue margined Context information box on the right side of the figure.

1.2 Context of this Thesis

When talking about context, the context of this thesis should not be left out from the discussion. This thesis is authored in the scope of the MediaSense project (see Kanter [13]), run by the ITM (Information Technology and Media) department of the MidSweden University, headed by Prof. Theo Kanter.

The objective of MediaSense is to enable context aware services in mobile sensor networks by utilizing protocols like Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (SIMPLE), Simple Object Access Protocol (SOAP) and Technologies like IP Multimedia Subsystem (IMS) and Java Media Edition (Java ME). The focus lies in the integration of sensor data, interactive media and context-sensitive communication. Mobile devices play an important role for MediaSense. They should be capable of acting as a context information mediator but also as a remote control, applying protocols such as Bluetooth, Global System for Mobile communication (GSM) or W-lan. Figure 1.2.1 shows the infrastructural road map for the MediaSense pilot-project, covering all major aspects of the system to be.

Previously theses conducted by Vidales, Swenson, and Chaz-alet ([14], [15], [16]) deal with the specification of a middleware-protocol for context distribution on an overlay network, which is based on the publisher-subscriber principle. Starting out from Context Exchange Protocol (CXP), a centralized approach, it evolved to be decentralized (Distributed-CXP). Prototypes for the protocols exist and are in every time improvement. The middleware essentially enables the subscription to context sources of interest. These sources could for instance be concrete temperature, location or humidity sensors, but as well user data, profiles or feeds (advertisement, warnings ...). Nodes in the overlay contain these concrete or abstract context sources. DCXP is a peer to peer protocol which is based on chord,
a distributed hash table scheme [17]. It synthesizes on top of User Datagram Protocol (UDP), providing basic quality of service features like reliable packet delivery.

Coming back to the MediaSense road-map, DCXP is situated in the heart of the P2P-tagged cloud (see 1.2.1), covering the context exchange between different parties. It enables the providence of services to diverse secured operator networks. The services themselves are defined on the server side in the conceptual IMS cloud in the upper left corner. This thesis discusses the blue margined context information box on the right side of the figure. A context-base, holding live and history context data for all MediaSense users should exist, enabling querying, context updates and reasoning. Standard protocols such as SOAP or SIP make them available via the networks and sub-networks. The context data may or may not be distributed across individual databases. The context-base is the source for retrieval and decision making in the architecture. A portfolio of multi-media files should be part of the picture as well. The tagging, consumption and sharing of multimedia data should be one of the key functionalities of the system.

An important driver for the concrete provision of the services to the (mobile) end users resides in the IMS-tagged cloud. IMS stands for IP Multimedia Subsystem and is a collection of specifications of the 3rd Generation Partnership Project (3GPP). It’s objective is the standardized access of services from within heterogenous networks. The key protocol for communication is SIP, which establishes connections for end users. In the MediaSense context, IMS addresses a concrete server, which enables the access to an IP sub-network, the Ericsson IMS. The upload of applications to this server enables service-consumption from within the mobile phone. Java Midlet’s (web applications) can be uploaded deployed and thereby made available to the mobile users.

An allocation of services requires permanent storage in order to deal with communication interruptions or disruptions in operations of mobile devices. That is where this thesis comes into place.

## 1.3 Problem Statement

The context data of information sources in a pervasive, decentralized network of things (e.g. in the scope of MediaSense) has to be acquired and archived as a basis for innovative context-aware applications.

A context-base enabling global availability of context data with respect to the environment (servers, connections, routers,...) or devices (capabilities, OS , available communication links,...) could be invaluable for decision making processes.

The key features for such a context-base are flexibility, extendability, searchability, protectability, openness and maintainability. Further more, it is not sufficient to store raw context data. There has to be a means to store meta data for context itself. Additionally, context data could be specified in various representations (e.g. de-facto vs. de-jure standard). An integrated mechanism for the translation between formats is crucial, because retrieval
and reasoning would otherwise not be enabled on the global scale.

The task is to find or create a context model that suffices the here mentioned requirements. An implementation for the model has to be developed and an API defined to allow access to the context-base. Further more, the context-base should provide out of the box support for optimized context retrieval in n-dimensional context-spaces.

1.4 Related Work

In C. Linnhoff-Popien et al [18], the authors critically analyse the applicability of certain context modeling approaches for ubiquitous computing. They proceed by comparing many concrete models referring to their depicted six key requirements:

- distributed composition
- partial validation
- richness and quality of information
- incompleteness and ambiguity
- level of formality
- applicability to existing environments

The survey discriminates six approaches for context modeling. It concludes that Object Oriented Models and especially ontologies in practise suite best for context modeling. However, this study does by no means include time complexity or the maturity of tools, supporting these approaches.

An interesting ontology based approach is presented in Korbinian et al [8]; the Aspect-Scale-Context (ASC) model. It’s core concepts can be seen in figure 1.4.2. The model aims to define precisely the meta data of context, defining concepts like aspects, scales and contextinformation. The meta model is specified in a way that “each aspect aggregates one or more scales, and each scale aggregates one or more contextinformation. These core concepts are interrelated via hasAspect, hasScale and constructedBy relations”. Where an aspect suits as ”any dimension of the situation space”, is a scale ”an unordered set of objects, defining the range of valid context information”. It specifies the valid domain for a concrete contextInformation instance of an aspect regarding it’s scale. This is a very general description and could be applied to whatever domain specific data, desired. An aspect could for instance be temperature. It’s scales could be FahrenheitScale and CelsiusScale.

The ASC-model can serve as a so called transfer model. A transfer model may be used ”... to employ the knowledge expressed in other context models.” The notion of IntraOperation’s allows for translations of contextinformation from one scale to another. The paper does not restrict the how IntraOperation’s are to be treated by the modeler. That way, transitive closure for translation with respect to an aspect can not be guaranteed. This might cause some problems, but makes the model very flexible and open.
Another feature of the model is the MetricOperation, which allows for two context information instances of the same scale to be compared with respect to its defined order. This is necessary because scales in the model are unordered sets. The ASC-ontology is a very solid meta-model for the integration of context information of any kind. No implementation of the ASC model had been found.

Uschold et al [19] present a thorough introduction into ontologies. One important aspect covered is the application of ontologies as so called Inter-Lingua for the integration of existing models. If following simple rules, it is possible to make semantically equal, but syntactically or metrically different concepts or values globally available for many stakeholders. More about that in chapter 2, Context Modeling.

In Kohvakko et al [2], the authors present another context modeling approach, based on the semantic web standards. The oeuvre gives a thorough introduction into the context modeling area. The research abstracts from implementation and technology issues and requires all parties to use the same context exchange protocol, which is defined in the thesis as well. A service architecture is proposed, rounding off the thesis.

JCAF (Java Context Aware Framework) [20] presents a more practical approach. It is a Java-based set of APIs for creating context-aware application. Because Java is the language of choice for MediaSense components, a thorough look into comparable efforts could be useful. Anyway, JCAF does neither bring along a means for persistence nor for context retrieval.

In Zaplata et al [21], the authors give a very fresh overview of existing context-aware general purpose models and their implementations. They further on focus rather on distributed process execution.

**Relevance**

There is a lot of research conducted in the field of context modeling. This short enumeration of context literature is by no means complete. The idea is to present varying, interesting approaches, that can be compared to some extend with the authors results, or served as sources for the authors ideas. The literature had been picked due to it’s relevance for context modeling in sensor networks and context modeling in general.
The focus of this work is to not only to contribute with a model, but with a concrete prototype. This prototype should use technology that suite context-modeling in environments where context could change dynamically and that rapidly.

The majority of approaches in this area uses the semantic web technologies. The common approach for persistence of context data are so called triple stores (more about that later in the document). These triple stores show significant problems with respect to scalability (see e.g. [22], [23]). The author thinks that an object oriented approach for design and persistence would better suite the stated requirements. This will be further discussed throughout the entire thesis.

Since OODBMS show impressive results in performance for complex data models and large amounts of data ([24], [25]), an application for context-modeling should be considered. In context-aware applications, semantic relations between individuals and things have to be defined and traversed frequently. Object oriented databases supply a graph model for the storage of class instances and referencing between the objects. Therefore they a-priori supply an optimized way of storing data for complex models with many references and relations. In a very recent study the worldwide leading consortium for performing scientific experiments, CERN (Organisation Européenne pour la Recherche Nucléaire), compared today’s database capabilities for the possibly largest database that should ever be build ([26]). Their conclusion was that the only, sufficiently scalable, choice would be an OODBMS.

Since the current open source object oriented databases are generally focussed on the embedded sector, they suffer from major drawbacks in multi-user support and efficient treatment of transactions (for instance db4o and NeoDatis). This thesis tries to recommend a hybrid solution. A platform that abstracts from concrete database structures is becoming applied to enable relational as well as object based storage of concepts. The platform fulfills the job to give the user a consistent interface, despite of concrete storage patterns.

The idea is to firstly apply a relational database under the hood, while waiting for the enhancement of object oriented databases. These databases already exist, but they are not open source (e.g. Versant Object Database or Objectivity/DB).

Because the ASC model has a significant influence on the CII-object-model, proposed in this thesis, it’s key features transfer model and metrics operations where important to be mentioned mentioned. The proposed model tries to combine the advantages from the two most popular and influencing model types: Ontologies and Markup-Scheme models. They, and their relation to the proposed model are further discussed in chapter two, Context Modeling.
1.5 Context on the Internet

Chapter 1.5 will give a brief introduction into how context is expressed in the semantic web. This is relevant due to the fact that the semantic web is the most commonly used approach in context-modeling, as mentioned further up in the document.

W3C

The World Wide Web Consortium (W3C) initiative is responsible for the definition of data representation and programming languages standards concerning the Internet. One of the biggest challenges for the Internet is how to formulate and organize the distributed knowledge in a way that it can be treated and interpreted appropriately with respect to its meaning, its semantics.

One example, which is often referred to for semantic retrieval is the 'burning bush'. Committed to a search engine, it should be interpreted according to the probable context and the semantic definition of the word 'bush'. Therefore, it should rather return results that are related to bible studies, than to terroristic acts against the former president of the United States. This is approached by the semantic web initiative, guided by the W3C consortium.

OWL

The W3C consortium recommends Web Ontology Language (OWL) as the knowledge representation language for the Internet. OWL is an in XML expressible language (see next paragraph for RDF(S)) that defines metadata or graphs, following domain specific ontologies, giving the semantic web entries a meaning. That OWL is based on description logic's enables the application of reasoners upon its models. Description logic's are a decidable (or computational complete) subset of first order logic's. OWL comes in three dialects: Lite, DL and Full. Their names are related to their expressiveness. Each dialect is an extension of its predecessor. Where Lite and DL are decidable, Full is not. Decidability and expressiveness of these dialects have much impact on the applicability to problem domains. Even though the Full dialect sounds tempting, reasoning upon its ontologies could result in infinite executions without the returning of any result.

RDF(Schema)

RDF is a language for expressing subject-predicate-object triple statements. It is the recommended standard of the W3C for the serialization of these statements. RDF-Schema is a knowledge representation language which main objective is to structure RDF resources. By that definition, these loosely coupled resources become connected. The main concepts of RDF(S) (e.g. subClassOf, domain, range) are part of OWL, which is more expressive and by this adoption made compatible to RDF statements.

1.6 Thesis Structure

The thesis consists of six chapters and three appendixes. This chapter, Introduction, gave a general overview of the topic, the problem statement and related work.
Chapter 2, *Context Modeling*, discusses the most relevant modeling approaches, before presenting the proposed CII-context-model.

Chapter 3, *Architecture & Implementation*, presents an architecture and the concrete implementation from a very high level of abstraction.

Chapter 4, *Context Retrieval*, explains the search and retrieval functionality, placed at the disposal by the implementation.

Chapter 5, *Discussion*, encloses the final discussion of the model with respect to the problem statement and existing work.

Chapter 6, *Conclusion*, sums up the results from the study and gives an outlook for future work.

Appendix A, *Use Cases*, formulates three problems for which the context-base could serve as a means for placing global context at disposal. Possible solutions, adapting the context-base, are presented. One of the scenarios, the traffic congestion scenario model, had been implemented using the CII-context-base implementation.

Appendix B, *Tutorial*, describes how to deploy the application, so to enable its services for the developer.

Appendix C, *Software Design & Implementation*, explains more generously the technology and the implementation from a developer point of view. Reading this chapter should significantly simplify the refactoring or extension of existing components.
2 Context Modeling

By the vast amount of publications and projects in the area it is observable that context modeling is a discipline which is seeking much attention. [18] compared 6 different approaches to context modeling in general:

- Key-Value models
- Markup Scheme models
- Graphical Models
- Object Oriented Models
- Logic Based Models
- Ontology Models

Before diving into their explanations, a general definition of ontologies will be given. This is essential because the term ontology is ambiguous in the sense that there is no universal consensus on the concrete requirements an ontology has to fulfill. The focus is here on ontologies because in some sense the other models could be seen or described as ontologies.

2.1 Ontologies

An ontology is a "formal, explicit specification of a shared conceptualisation", Uschold & Gruninger state in [19]. This shared conceptualisation or understanding can help solving communication problems between people, organizations, and software systems. Having parties, coming with specific viewpoints and diverging or possibly overlapping concepts, lead to 'poor communication.' Further on this leads to 'difficulties in system specification'.

The application of disparate methods or languages used for modeling leads to poor reuseability and interoperability, what clearly indicates wasted effort. The application of an ontology forces a shared understanding to solve these problems. Figure 2.1.1 gives an overview of what topics ontologies tackle. But how do these ontologies look like? "An ontology necessarily entails or embodies some sort of world view with respect to a given domain. The world view is often conceived as a set of concepts (e.g. entities, attributes, processes), their de nitions and their inter-relationships this is referred to as a conceptualisation." [19] These concepts and relationships can vary in representation from highly informal (e.g. spoken language) to rigorously formal (formal semantics, axioms, theorems supported by proofs of important properties like completeness or correctness).
After reviewing the term ontology, each method for representing concepts and their relationships could be considered as one. There is little consensus upon what mandatory parts have to be present in an ontology. Atkinson [27] discusses the differences between models and ontologies. It employs the point that all ontologies are a subset of all existing models, but the membership is not simple to be defined. This thesis sticks to the interpretation that each context model providing concepts for concepts, attributes, correlations between concepts and inheritance is considered an ontology. However, some ontologies might be more expressive than others.

2.1.1 Inter-Lingua

When assimilating syntactically different concepts with the same or similar semantics, a translation is required for the enabling of their contents from and to one another. A pattern for the generic resolution of that problem is used in the proposed CII model. This pattern is referred to as Inter-Lingua. Inter-Lingua describes the need for one central representation, which all parties accept as such. Each party has to specify a translator from it’s models concepts to the inter-lingua concepts and vice versa. That way it can be guaranteed that a valid translation from each model to another exists.

As a consequence only two translations have to be specified for each concept, resulting in a total of $O(n)$ translations for $n$ concepts. This is much more efficient than having a translation for each pair of concepts, which would require $O(n^2)$ translators to be specified. Figure 2.1.2 visualizes this principle.

2.2 The Approaches

For simplicity all in C. Linnhoff-Popien et al [18] mentioned models, except from the key-value models, are specified as ontologies. They don’t express just simple taxonomies, but rather advanced concepts for the specification
Figure 2.1.2: The purpose of this figure is to show the growing system complexity if no Inter-Lingua is used for translation (left side). [19]

of concepts and relationships.

Key-value models are not considered ontologies. They are very simple hash-map like structures, suffering from the "lack of sophisticated structuring for enabling efficient context retrieval algorithms" [18].

The other extreme are the logical based models. They are very formal and able to represent very complex context structures. The problem with logical based models is often that a) their learning curve is very flat, b) it exists no good tooling to support the usage and c) their concepts don’t really match the patterns of languages used in today’s enterprise development (Java, C# . . .). Bridges to formal and logical languages like F-logic or Prolog have to be used as mediators.

The attentive reader notes that the discussion is not only about the general model, but as well the applicability of a concrete implementation. Even thought modeling is a conceptual step, the decisions here can have a large impact on the implementations feasibility in the developmental phase.

The interesting modeling approaches are the other four; enumerated in the beginning of this chapter. Ontologies with their concepts and instances have much in common with the object oriented approach and it’s classes and objects. What makes the ontological approach special is the fact that it explicitly uses the notation for ontologies and that it’s concepts are very flexible in their interpretation.

The graphical model of main attention in the modeling world is UML (Unified Modeling Language). It is the standard for modeling of any kind of static and dynamic behavior for software systems. The connection to object oriented models is strong.
Figure 2.2.3: The stairs are showing the correlation between certain data representation models and their classification with respect to interoperability. [27]

Markup-scheme models are generally exemplified by XML (Extensible Markup Language) structures. They are tree like, holding a root and sub-nodes which themselves can be the root for sub-nodes and so forth. Their usage suits great for the platform independent exchange of data between heterogenous devices, programs or operating systems.

Figure 2.2.3 tries to depict the approaches expressiveness with respect to semantics. Starting out from only syntactically interpretable models, this enumeration goes all the way up to very expressive models like modal logic’s.

The applicability of more expressive models is increasing steadily. This is caused by expanding communities, creating tools and frameworks to define and use them. Anyway, there are still a) significant performance problems when it comes to conceptual models, and b) a lack of large scale education/understanding of concepts; in particular the ones based on logical theory.

After reviewing the different ontologies, it would be useful to combine the best from the two worlds of markup-scheme languages and what is here referred to as conceptual models (Graphical, Object Oriented, Ontologies) in one approach. A union of markup schema exchangeability, the universality and extendability of ontological models, the visualization of concepts by UML and the compatibility to programming languages by object oriented design should be inspected. This goal was attempted by the CII-model and it’s implementation, proposed in the next section of this chapter.

2.3 The CII(Context Information Integration)-model

The CII-model is an ontology based on the conceptual models specified in the former section. Context is expressible and exchangeable by any sort of
Figure 2.3.4: The CII context model, represented as a UML class diagram.

serialized syntactical construct, for instance XML trees.

Figure 2.3.4 shows the concepts in a Unified Modeling Language (UML)-class diagram notion. Correlations between classes have so called cardinalities. Similar to the usage of regular expressions, * stands for any many. For instance 1..* means that at least one instance of the class has to exist for each correlation. Only 7 concepts describe the whole context-ontology. The following enumeration describes the meaning for each of them:

**InformationSource** InformationSource’s are the things in the model, holding the current context. Each InformationSource holds one value at a time; an instance of ContextValue. A temperature sensor could be an InformationSource.

**ContextValue** ContextValues represent context information in the model. The entry `<temp value='30 degrees'>` could be such a context-value for a temperature sensor.

**Entity** Entities represent the things in the model that context can be attached to. Each Entity can be related to other Entities in two ways: higher or lower in a hierarchy. These relations are in general of aggregation or compositional type. They could represent social networks between Entities (e.g. cascaded group memberships). One such example is presented in 2.3.5. This should not be mixed up with inheritance, where one Entity is parent of another. An Entity has any desired amount of InformationSources attached to it.

**Dimension** A Dimension defines a domain for context information. It therefore specifies what ContextValues are valid, and which are not. A Dimension for a temperature ContextValue could be celsius, spanning over the domain $\mathbb{R}$. 
A Representation is a template to present context information. Every InformationSource holds a ContextValue of one specific representational type. Beyond that, each Representation follows an Aspect. One possible Representation could be called xml_celsius, representing ContextValues like this: `<temp value='X degrees'>`. X stands here for the variable of a Dimension.

Aspect Aspects cover the kind of context that can possibly be expressed in a model. Such an Aspect could be temperature. Each Aspect has any desired amount of Dimensions and Representations, requiring at least one of each. One particular Representation has to be specified as the standard Representation for each Aspect. It serves as the inter-lingua (explained in 2.1.1).

RepresentationMapper A RepresentationMapper is responsible for the translation from one specific Representation of an Aspect to the standard Representation (mapToStandard) and vice versa (mapFromStandard). Let’s consider two Representations xml_celsius and simple_fahrenheit for the Aspect temperature, where xml_celsius is the standard Representation. The RepresentationMapper for simple_fahrenheit has to supply the two functions

\[
\begin{align*}
 f_{\text{map}}(x_{\text{simple_fahrenheit}}) &= y_{\text{xml_celsius}} \\
 f_{\text{invert}}^{-1}(y_{\text{xml_celsius}}) &= x_{\text{simple_fahrenheit}}
\end{align*}
\]

The RepresentationMapper ensures the transitive translation closure for all Representations of one Aspect. That means, the model itself is able to translate between all types of Representations for each Aspect. This is a key feature for the integration of context in different formats and units.

AspectComparator The AspectComparator is no obligatory part of the model. In case an Aspect specifies an order for it’s ContextValues it could be applied as a means to determine the order correlation or distances between two values. It is therefore possible to specify one AspectComparator for each Aspect. The idea behind the AspectComparator is that the CII-model, without knowing anything about the Aspects or context contents, is capable of optimizing it’s retention. The AspectComparator provides two functions. One to compare two ContextValues, returning their order regarding to a documented metric. The other one returns the relative distance between two ContextValues with respect to that metric.

2.3.1 How Context is expressed

ContextValue, Aspect and Representation serve as mediator concepts for context. The real essence of the model is to give Entities contextual meaning. It is important to tie the structures of the CII-model to their contextual meanings, so to receive a correlation between syntactical constructs and their semantic interpretation. The following definitions attempt that:

Definition 2. The context of an InformationSource at a moment in time is defined by it’s ContextValue.

Definition 3. The simple-context of an Entity at a moment in time is the set of all attached InformationSources contexts.
Figure 2.3.5: Instances of Entities and their attached context, as well as their hierarchical relationships.
Definition 4. The context of an Entity at a moment in time is its simple context including the simple-context of all Entities in its hierarchical proximity. How proximity is defined, depends on the eye of the beholder.

The context of an Entity is not expressed entirely explicit. Context very much depends on the subjective interest of the prospect. That is why context is partially implicit in the CII-model.

The model viewed at a moment in time always represents a snapshot of the current overall context. It is possible to extend the model, expressing history context for InformationSources as well. When extending the model for history context, the 1\(\rightarrow\)1 correlation between InformationSource and ContextValue becomes altered to be 1\(\rightarrow\)*. Timestamps mark the ContextValues, so for each value it can be derived in which time frame it had been valid.

### 2.3.2 Subconcepts and Inheritance

The CII-model provides the modeler with a means to express context for abstract things like InformationSources or Entities. These concepts are so general that their semantic interpretation is very vague.

By the definition of sub-concepts, concrete domain specific Entities like for instance Person, Car or Club could add semantics to the model. So the CII-model itself is just a template for the definition of domain-models and their sub-concepts. In the fashion of an upper-ontology, it defines the boundaries in which the domain-models may evolve. A domain model example for a traffic congestion system can be found in appendix A.
To bring the reader intuitively closer to the way context is expressed, figure 2.3.6 presents the conceptual model with exemplified entries. This visualizes the relationships, previously mentioned, in a more obvious way. However, it is rather static. The Entity, InformationSource network in 2.3.5 with concrete sub-concepts could dynamically change over time.
3 Architecture and Implementation

Having a suitable model for the sensor networks context is very useful when investigating possible approaches for how to persist and expose context information to context-aware applications.

This chapter proposes a context-base architecture, which is independent of technology. It discusses how context could be enabled as a service, applying a middleware of choice. For that reason an API is proposed, covering the CII based functionality.

The architecture is followed by a prototype written in Java, applying tools like Apache-Ant, DataNucleus and PostgreSQL Relational Database Management Systems (RDBMS).

A thorough description of the software design can be read in Appendix C Software Design & Implementation. The object model, database schema and data flow are discussed there.

3.1 General Architecture

A software architecture describes the fundamental components of a software system and their interactions. Where the software design itself can be very detailed, an architecture lies it’s focus into the presentation of the general picture.

The decision for a layered architecture is motivated by the commonly applied n-tier approach architectures in business development, where at least persistence, business logic and presentation are encapsulated. Due to the fact that the context-base can be seen as a middleware that abstracts the persistence from context consumers/providers, responsibilities are simply defined via layers. The proposed architecture is visualized in figure 3.1.1. The further up the layer, the closer it comes to the end user. A description with obligations for each layer follows:

Context Aware Application  From the perspective of a context-aware application, the context-base is a rich service for the management of context and context models. The applications could be a) consumable by end users or b) perform automated, context related tasks. The developer of context-aware applications either reuse existing domain-model concepts from the context-base or extend it’s model to suite his needs by adding concepts himself.
Figure 3.1.1: The 5-layer architecture of the context-base. Starting at the bottom with the data persistence, ending with the service consumption.

**Context-base API**  The context-base API provides the context-aware applications with standard interfaces, forming the only valid access to the context-base. These interfaces have to contain methods for querying and altering context, as well as to extend the context-base’s concepts. These domain extensions could contain Aspects and their Representations or new sensors or sub-concepts of type Entity (e.g. Vehicle, Group ...).

**Context-base Controller**  The controllers task is to handle incoming API-based commissions and act or sense upon the context-base. It therefore encapsulates an implementation of the context-base API. It is desired to be thin and stateless to enable the distribution of multiple controllers, accessing the global context-base concurrently and returning the same results, while offering higher scalability.

**Access Platform**  The access platform’s job is to abstract the concrete context data storage from the context-controller. It should guarantee data-consistency and concurrent access by the application of the transactional paradigm. An access platform in general offers different options in what way data is stored. Examples are RDBMS, Object Oriented Database Management System (OODBMS) or Extendable Markup Language (XML) files. Because it embodies all related mappings, a migration from one mechanism to another would be rather smoothly conducted. This allows for simple comparisons of different datastores with respect to performance and scalability.

**Physical Context Store**  The physical context store is where the concrete context data is stored. There is a common distinction between transactional data, which is frequently updated and in all time use and historical data which is not becoming updated and just grows over time. The first type is often referred to as Online Transactional Processing (OLTP) and the second as Online Analytical Processing (OLAP). If these concepts are treated within one physical database or spread over various databases depends on the context-base usage and will be further discussed in (3.3.1).
3.1.1 The Context Controller

At the heart of this architecture is the context-controller. It is the concrete achievement behind this thesis. The controller is responsible for context access, modification, extension and validity in the context-base.

Figure 3.1.2 shows the controllers sub-components. For a context-aware service, it has to be verified if the existing model comprises all desired model extensions. If that is not the case, the model can be extended via the usage of the context-base API. The context Model Integrator should deal with related tasks.

The creation, deletion and alteration of the meta-model should be managed by the context-controller as well. A mechanism to administrate and maintain the database schema has to be given. Resilience and robustness are very important to be treated; flawed model extensions should have only local impact and not harm the overall usage of the context-base. These tasks fall into the responsibilities of the context Model Creator/Maintainer.

Model Integrator as well as Model Creator should make use of the context-controller’s retrieval engine. The retrieval engine is the general channel for querying context information from the access platform. It provides build in functions for a simplified and time optimized context retrieval. This module should be exposed to the context-aware applications by the context-API.

3.1.2 Service Enabling

For a smooth integration into existing applications, it should be possible to use common technologies, like Web services. Web services enable the interoperability of applications, written in different programming languages, by exchanging XML messages via standard protocols like Simple Object Access Protocol (SOAP) or Representational State Transfer (REST). Since these application layer protocols are XML based, they can be simply used via one
of the most frequently used Internet standard protocols Hypertext Transfer Protocol (HTTP).

Since the context-base is preferably developed in the Java programming language, the enabling for distributed computing via frameworks like Remote Method Invocation (RMI) or Apache River (former Jini) are other paths to go. An (incomplete) enumeration is depicted in figure 3.1.3.

The advantage of in Java embedded distributed technologies is the direct treatment of classes and objects. The CII-model allows to extend it’s concepts. If a proposed implementation defines all concepts as classes, the context-base has to offer a means to store them in a way that they can be shared between context-aware applications. This implies that runnable code could be exchanged between these applications. Concepts themselves could encapsulate different storable services.

The red marked entries in 3.1.3 are not out of the box capable of delivering the stored services or methods to the end user. The forced serialization narrows the functionality to context exchange and persistence; code can only be executed from within a Java virtual machine.

DCXP for instance serializes all context resources into so called Unified Context Identifier’s (UCI), which are special purpose Uniform Resource Identifier’s (URI). The idea behind URI is to address resources in a standard way and transparently to the user, regardless of it’s actual location. The context resource addressing towards the context-base simply requires a syntactical construct for the representation of Entities and InformationSources. Anyway, for the moving of code, a more sophisticated integration into DCXP would have to be developed. However, by far not all applications need to move code for the effective achievement of their purposes.

So, by applying Java technologies it is possible to reuse code from different context domain models, executed on any JVM, having access to the context-base. That means the context-base contains a way to store concepts, but as well concept related code; and has therefore the ability to store services, covering Turing completeness. This might not be a must for all applications but could be used for reasoning and agents in the system, accomplishing automated tasks, while operating on global context.

### 3.2 Application Programming Interfaces

From an application centric perspective, users can be clustered in two overlapping groups: Context providers and context consumers. Where providers create domain models, reuse existing model extensions and by that modify the schema itself, consumers only query, add, modify and remove context. Hence, consumers only request or change the context payload itself. Context providers supply the consumers with services/applications that are to some extent context-based or context-aware. The API is split up in two modules with respect to this classification. What concrete functionality they provide to the user will be discussed in the following subsections.

For application developers, keenly interested into getting started, it is rec-
ommended to read the appendixes A and B and as well take a look at the thorough explanation of functionality in the JavaDoc for the developed modules. In here, the author only briefly goes through the methods that area of main interest.

An interface for the management has not been defined. The context-base should to a large extend manage itself. The main administrative functionalities should be the initial creation and final deletion of the context-base.

3.2.1 Context Provider

The context-provider interface IContextSpecification and it’s signature can be seen in figure 3.2.4. Not every user should have access to implementations of this interface; only service/application developer that are authorized to extend the context-bases context model.

uploadClassToContextBase(conceptId, javaClass, jdoMetaDescriptor) allows the upload of model extensions at runtime. Like discussed further up in the document, the semantic expressiveness of the model depends on it’s model extensions. This particular method concentrates on the specification of Entity and InformationSource sub-concepts. The context framework requires three things from the extender. First, a globally unique id conceptId for the concept to be uploaded. Second, a Java class representing the concept for the storable instances. Third, a descriptor which contains the how and the what will be stored of the concept, specified by the Java Data Objects (JDO) standard. Despite semantic extension, this approach has two additional advantages for the implementation. Queries can be optimized in precision and performance when sub-concepts are used as filters.

uploadClassToContextBase(conceptId, javaClass) is similar to the previous method, but has the focus on concepts for the context-base that should not be stored. In particular, implementations of RepresentationMapper and AspectComparator are addressed by this method. These concepts are obligatory, since they help organizing ContextValues in a unified way. To apply this generally, reflection techniques are used frequently to instantiate, access and manage unknown concepts at runtime.
Figure 3.2.4: The model extension interface, used by all applications modifying or extending the context model.

Figure 3.2.5: The consumer interface, used by all applications requesting or updating context related data.
3.2.2 Context Consumer

The context consumer interface IContextConsumption with it's signature can be seen in figure 3.2.5. Service developers should use this interface for the treatment of real context data. A description of the methods follows:

requestContextInformation(jdoqlQuery) The search engine processes queries based on JDOQL, which will be further discussed in the following section. Queries are firstly forwarded to the access platform and the respective result set is back-propagated to the calling instance.

translateContextValueToRepresentation(contextValue,fromRep,toRep) This method translates the ContextValue expressed in Representation fromRep to Representation toRep. That might include unit conversions and changes in markup. The externalization of this method does allow the integration of context expressed in different formats not only on the context-controller layer, but as well directly on the service layer.

getEntitiesByContext(contextFilter[],conceptFilter,refPoint) provides the user with a sophisticated mechanism for the retrieval of Entities, based on their context. Three things are to be specified for retrieval. First, the context filters by which matching Entities are selected. Each of them represents one Aspect and filters by criteria based on it's dimensions. Second, the conceptFilter is a sub-class of Entity. In case it becomes applied, a significant performance gain is to be expected. The third thing to be specified is the refPoint. It wraps a value in standard Representation of an Aspect, according to which the result-set array will be ordered by. If the reference point is null, the result will not be ordered.

getDistanceForDimension(aspectId, contextValue1, contextValue2, dimension) returns the distance for two instances of a certain representation type, regarding a specified dimension. For that it uses the metric which is defined by it's Aspects AspectComparator implementation. The result is expected to be normalized, so it gives the calling instance a means for comparison to other distances. This method could be especially interesting for agents that don’t know much about the domains, but want to sense if Entities are closer or further away from each other with respect to a Dimension.

getDistance(aspectId, contextValue1, contextValue2) is very similar to the previous method. The difference here is that it does not only compute a one dimensional distance, but rather an n-dimensional distance, depending on the metric that is specified in the respective IAspectComparator. For a location on a two dimensional map, it would not only return the distance with respect to the x-axis, but with respect to the y-axis as well. Possible metrics here are euclidean and Manhattan.

3.3 Implementation

The context-base implementation follows the previously presented architecture. The guiding picture is figure 3.3.6. It shows what technologies are applied to the abstract layers. Since the programming language of choice is Java (version Tiger 6.0), the supporting frameworks are all written in Java.
as well. The applications to be could range from Graphical User Interface (GUI) integrating web services to self-aware agents automatically running as background processes. Midlet based applications that enable user services on mobile devices might be of special interest to the mobile community. However, no assumptions are made regarding the service platform itself.

The practical contribution of this thesis lies in this prototype. In focus is the development of it’s context controller, including the externalizing of useful features from the access platform in an application developer friendly manner. Access Platforms to choose from where for instance Hibernate[28], ibatis[29] or DataNucleus. The choice fell on DataNucleus because among other things it is the only one explicitly allowing ODBMS as an option for persistence. How exactly DataNucleus can be used for context storage and propagation will be discussed here.

### 3.3.1 The DataNucleus Access Platform

DataNucleus is an open source project that "provides products for the management of application data in a Java environment" [30]. The big advantage of applying such a technology is "that you don’t need to take significant time in learning the oddities of particular data-stores, or query languages and in-

---

Figure 3.3.6: The implementation related architecture of the context-base. Concrete concepts and relations are explained by the arc descriptions.
Figure 3.3.7: The DataNucleus platform with it’s two central components for data access and analysis. In the scope of this thesis the access platform is of main concern (figure from http://www.datanucleus.org/).

stead use a single common interface for all of your data” [30]. It deals with the problem of impedance mismatch between flat relational database tables and complex object models.

Picture 3.3.7 shows the aspects DataNucleus covers. The application of DataNucleus in the context of this thesis is centralized on the blue box, tagged Datanucleus access platform. In ongoing work, discussions about data mining and warehousing may come up. The DataNucleus analysis platform would be probably one of the first options to examine. One of the key drivers of DataNucleus is that it enables distributed access. Distributed access simplifies the decentralization of applications. DataNucleus and it’s integration goes under the Apache 2 licence. It is open source and there is no charge to be payed for the usage.

Usage

DataNucleus suites this thesis needs, because it provides a means to use a standardized retrieval language while keeping the possibility to use different database management systems, depending on setup, requirements and resources.

The DataNucleus access platform deals with any different type of persistence concerning Java objects. In our scenario it uses a Java Database Connectivity (JDBC) connection to a RDBMS. The concrete publisher (e.g. Mysql, Postgres, Derby…) can freely be chosen by the administrator. The platforms schema tool provides a flexible support for the ad-hoc integration of classes and modification of database tables for relational databases. By uploading a concept, which is represented by a Java class and a JDO descriptor, immediate integration takes place. The concepts are directly made
available to all applications. The schema tool can be used by its respective Apache-Ant tasks, that are invoked from within the context-controllers Java source code.

In theory, it would be already possible to use an ODBMS. The access platform does in general support some of them, for instance db4o (Database for Objects) [31]. The current problem is that the implementation of its DataNucleus adapter is flawed, and therefore not usable (see http://www.jpox.org/servlet/jira/browse/DBFO-27). The module was developed in the scope of a student project. A verification from the DataNucleus side showed that it contains flawed attach/detach behavior for permanently stored objects. That causes the access platform to not react rightly on general single-line JDOQL queries. When this is fixed, a direct enabling of db4o would be possible. This would probably brings along a significant win in performance.

**Live and History Data**

The general idea of discriminating live and history data is that they are used for different purposes, due to their nature. History data is often a very good foundation for reasoning and discovery of new knowledge. Tasks that are computationally intensive and only involve read activity are typical. The research within data mining is concerned with related analysis and knowledge extraction. The term data mining is somewhat misleading, since it does not deal with mining for data, but for information or knowledge.

Live or transactional data is generally added, modified and/or becomes deleted. In many scenarios these changes should be done fast and within short transactions, often involving user feedback.

The live data is stored for each InformationSource together with other InformationSources of the same Aspect type. The history data is narrowed at the time being to an extra table with data entries and timestamps, for each InformationSource. This table follows a unified name pattern, so it can be queried. The name structure is '${isid}.History', where ${isid} is the id of the InformationSource of interest.

The recording of history data does for this thesis only concern the Context-Values over time and not their relations to the dynamic Entity network. For instance the history of temperature values for a sensor is recorded, but not to whom it was attached to for each context update over time.

Even thought the code-base uses the concepts of real time context and history context, there is no clear distinction in terms of processing. However, because context in the past can be of importance for the interpretation of current context, it should be treated similar to the real time context in some sense.

So even though the architecture in general conceptually distinguishes between live and history context, for the prototype both are stored in the same physical database, even though in different tables. This approach has advantages and disadvantages, which will be further discussed in the Discussion.
3.3.2 Proof of Concept

A proof of concept was conducted to make sure that all desired features are supported for the transparent integration of the CII-model into a Java based, object oriented environment. The problem statement to be answered was:

Suppose there is a context-base C and context-aware applications A and B, running on two different remote machines. If application A extends the global context model by a class CL, is application B directly able to use class CL for the creation or retrieval of objects in the global context-base?

The setup contained three PC’s that were connected via an Ethernet link. One of them, running the context-base, the other two running application A and B respectively. The middleware of choice for the communication was RMI, a client server based approach, which allows the invocation of methods from remote objects in a location-transparent fashion. The communication is therefore not protocol, but API based.

For the verification, a very basic context model was created. This model contained a class called Entity, that holds an attribute id. By remote method invocation, a subclass SubEntity with an attribute name should be uploaded from the client A to the server C. This class should at runtime be integrated into the storable model. One instance of the SubEntity type and one of the general Entity type are then stored to the context-base by the remote method call addEntity on client A. This integration at runtime should enable the context retrieval for the newly added attribute name of class SubEntity. The following query

"Select from Entity where instanceof(SubEntity) && name=='test1'"

was send as a remote call of requestContextInformation from B. This should return only the one particular object of type SubEntity.

For the persistence, a db4o database was applied. After the knowledge about runtime class enhancement was acquired, a prototype using Ant scripts became created. The concept was verified to work. The prototype can be downloaded from the MediaSense Subversion server (https://193.10.119.42/MediaSense/repo/ContextBase/Prototype).

This proved that it is generally possible to extend the storable class model at runtime by subclasses, even containing further specialization. These classes are immediately retrievable and objects of that type are storable to, and retrievable from the context-base. This result shows the possibility of applying the approach to extend the model from which ever application, having access to an implementation of the context-base, even if they reside in other (virtual-)machines than the context-base itself.

The prototype showed as well that this accounts not only for subclass fields but as well for methods. It is therefore possible to store arbitrary java code inside of methods into the context-base, retrieve instances of that specific
type by JDOQL queries and invoke them from any application. This feature enables executable entities in the system, that themselves carry storable context.

3.4 The Context Controller

The context-controller prototype makes use of Apache-Ant build scripts for the creation, deletion and alteration of context-related data facilitated by the access platform. It considers and works with an object oriented model, delivered by DataNucleus, applying the JDO specification. Therefore, querying is done by object oriented queries, upon the object structure of the model. The language of choice is JDOQL, which is the standard querying language for JDO, being used for technologies like Entity Java Beans (EJB) in distributed, object oriented, systems.

Appendix A, *Software Design & Implementation* contains plenty of flow diagrams, depicting the processing within the context controller.

The Code-base

The context-base module that deals with the Java classes is the code-base. It could be considered a commonly shared extendable domain model library. Figure 3.4.8 shows how it fits into the big picture, involving applications and the context-base. By keeping this architectural structure, each application can take advantage of all concepts.

It is not only possible for the application developer to add domain-model-concepts to the context-base, but as well to utilize these concepts with Turing powerfulness. Since the concepts are coded as Java classes and uploaded to a shared code-base, this enables a vast amount of possibilities. One of them is to add further attributes to the concepts, for instance a global field `address` for a sub-concept of entity, `Person`, which is permanently stored, and enabled for retrieval by JDOQL. That means when this concept is added a) objects of that type can be created and stored and b) it can be referred to in queries. The developer has very much impact on what and how context is stored in the context-base.
Figure 3.4.8: The current solution for code sharing in the model. The domain model extensions (code) and API are sitting on a central location. This model should in future work be further decentralized.
4 Context Retrieval

Chapter 4 gives an insight about the implementations retrieval opportunities. Even though this thesis proposes object oriented persistence for context, other approaches should not be left out of the discussion.

Within the Language section, a short enumeration of possible approaches will be given. It finalizes with a presentation of the retrieval language, chosen for the purpose of this thesis, JDOQL.

Reviewing the definition of the CII-model, the simple context of an Entity was defined by the values of it’s attached InformationSources. So, the simple context domain could be seen as an $n$-dimensional context space. This $n$ is non static and can change over time, since it depends on amount of currently attached InformationSources.

The framework should offer a retrieval function that is able to filter Entities according to a subset of $k \leq n$ dimensions, so to find Entities that are close to a respective $k$-dimensional reference point. A discussion about how this feature could be a) implemented and b) made available to the user follows in section Nearest Neighbors.

4.1 Language

Any organized data representation requires a structured querying language, to retrieve, add or modify the contents in a unified and simple way. In general querying languages are of declarative nature. This means that they, in contrast to procedural languages, not force the user to describe how a result should be retrieved but only what it should contain. The quality of a declarative querying language can be measured by how user friendly and intuitively it can be applied. The query optimization with respect to their execution order should not be of user concern, if the user does not explicitly want to define it himself. Existing Approaches enumerates some of the most influencing querying languages in use. The subsection JDOQL presents the language of choice for this thesis.

4.1.1 Existing Approaches

There are very different approaches to model and retrieve data from within a data-source. These approaches very much depend on the representation and structure of the data. The querying languages for the commonly used data structures are enumerated here:
Structured Query Language (SQL) is by far the most widespread querying language. It assumes a relational data representation (in general a RDBMS) and facilitates creation, deletion and modification of schemas and their contents, as well as access control and meta-data management. SQL is a declarative querying language, applying the predicate calculus. Due to it’s popularity, all sorts of tools exist to enable data retrieval, warehousing and mining. SQL semantics are, however, not Turing powerful, meaning that not all possible computational tasks are executable by the language.

XPath & XQuery is the duo, which is commonly used for the structural querying of contents in XML trees, defined by schema specification languages like Document Type Definition (DTD) or XML-schema. Where XPath is simply selecting nodes by navigating through trees, XQuery is as powerful as SQL in it’s semantic expressiveness. XQuery uses XPath under the hood for the navigation within XML structures. This querying type is associated with the markup-scheme models, discussed in chapter two.

SPARQL (recursive acronym: SPARQL Protocol and RDF Query Language) is the recommended querying language for the semantic web. The semantic web bases on Web Ontology Language (OWL), which is the recommended way for authoring ontologies. In the general case, these OWL ontologies are serialized by Resource Description Framework (RDF) meta-data models, which in return are all instances of an XML-schema. These models are not more expressive than Entity/Relationship models or UML class diagrams, but due to their basis on XML, the exchange, parsing and interaction of data over the web is simplified significantly.

Knowledge representation and (or) ontology languages such as F-Logic or Object Querying language (OQL) are complex querying languages for object oriented data models. They provide means for e.g. the expression of object identities, inheritance, polymorphism, encapsulation and in query computations. Where there exist implementations for F-logic (e.g. FLORID\(^1\) for C++), OQL, which is the recommendation for object querying by the Object Management Group (OMG), had never fully been implemented. Anyway, OQL had much impact on the language chosen for context retrieval in this thesis, JDOQL.

4.1.2 JDOQL

JDOQL stands for Java Data Object Querying Language. It is seeking influence from SQL and OQL, being the standard querying language for JDO annotated Java classes. It is the querying language applied to the prototype. It’s syntactical similarity to SQL and semantic integration of object oriented concepts from OQL makes it a powerful retrieval language. Concepts like object identities, polymorphism, inheritance, variable specification and many build in methods, inspired by SQL (avg(), min(), max() …) make it a choice for direct querying the object model, retaining the application structure and avoiding impedance mismatch problems to a relational database.

However, some restrictions are given when comparing to OQL. The call of

\(^1\)http://dbis.informatik.uni-freiburg.de/index.php?project=Florid
methods from within a query is only possible for predefined standard queries on abstract data-types like for instance lists, sets or collections. This influences the way object fields are accessed. Generally, fields in objects should be private and only reachable by the calling of for instance setter or getter methods on the object. This is one of the key principles of object oriented programming and is called encapsulation. In JDOQL, fields are accessed by overcoming encapsulation and directly navigating in common ‘.’ delimiter notation. For a class with name ClassName, having a field fieldName, the field access would be conducted by ‘ClassName.fieldName’.

Query filters are applied in a OQL way, which reminds on UML’s OCL (Object Constraint Language) notation. The general JDOQL syntax follows here:

```
SELECT [UNIQUE] [<result>] [INTO <result-class>]
FROM <candidate-class> [EXCLUDE SUBCLASSES]
WHERE <filter>
[VARIABLES <variable declarations>]
[PARAMETERS <parameter declarations>]
[<import declarations>]
[GROUP BY <grouping>]
[ORDER BY <ordering>]
[RANGE <start>, <end>]
```

Because JDOQL is a pure retrieval language, the creation, modifications and deletion of contents is not supported. Since this thesis does not try to cover an introduction into JDOQL, see http://www.jpox.org/docs/1_1/query_jdoql.html for a more thorough documentation. A query example can for instance be found in Appendix A, Use Cases.

Since DataNucleus supports RDBMS under the hood, it contains a module for query translation from JDOQL to SQL. That means, DataNucleus accesses the database via the standard JDBC interface and SQL queries. It is possible to use SQL to directly query and alter the database schema, if a RDBMS is in use. However, it is recommended to use JDOQL for querying, since it allows the migration from a RDBMS to a OBDMS without any query or code changes.

### 4.2 Nearest Neighbors

For context-aware applications, an important question to be answered is which context information is of interest. Since context is a very subjective matter, the context engine has to provide a means to search the context neighborhood. Entities within a certain close distance to a reference point could hold valuable context information.

A related problem in computer science is the Nearest Neighbor problem. The problem is defined by finding the neighbors to an entity with respect to certain conditions. These conditions could be seen as an n-dimensional space of context values. In particular the k-NN (k-Nearest-Neighbor) problem is of interest here. It specializes the problem to the instances where
\( k \in \mathbb{N}, k \leq n \) nearest neighbors with respect to an n-dimensional condition-space are requested.

The k-NN problem is NP-complete. That means there exists no solution in the bounds of a polynomial time-step-function. This implies that the time to solve an instance of this problem grows exponentially by the amount of context entries in the system. As a consequence retrieval becomes very slow. In return, this means that the scalability requirement becomes violated, what implies that the applicability of the algorithm is not given for many domains; especially regarding real time decision making.

To overcome this problem, some approaches could be taken. The general idea is here to simplify the problem. The problem could for instance be reduced to only a certain amount of nearest-neighbors, that can be requested by the user (for example \( k = 5 \) or \( k \in \{3, 5, 7\} \)), so that data can be pre-organized at storage time for all queries in general.

The same idea could be applied to the n-dimensions, where a threshold of for instance \( n = 3 \) could mean that a maximum of 3 filters may be combined for a query.

Both mechanisms make the implementation very problem specific, and therefore influence reuseability and interoperability negatively.

Another approach is to loose up the deterministic condition that the exact k-NN have to be retrieved with certainty. Probabilistic algorithms could deliver the k-NN with a certainty or error-percentage of containing wrong entries, enforcing a higher performance. In domains where certainty is significant, this can not be applied.

The presented ideas are all non suitable for this thesis. The service has to deliver results with certainty and that within certain time limit corresponding to real time requirements. The here proposed solution does that, when considering logical time steps.

Nearest neighbor search could be simplified to searching in ranges of the n-dimensions. The user must know himself how far away entries of interest may be. The flipside of this coin is that bad guesses could result in either far too many matches or possibly none at all.

Existing solutions for solving k-NN like Kd-trees\(^1\) or probabilistic approaches such as LSH (Locality Sensitive Hashing\(^2\)) will not be delt with in this thesis. They simply don’t scale for the real time problems to be solved.

### 4.2.1 Searching In Ranges

For the MediaSense project, the context-base does not know in advanced

1. what kinds of Aspects will be filtered,

2. what order (if any) the Aspects Dimensions define.

\(^1\)http://en.wikipedia.org/wiki/Kd-tree

\(^2\)http://www.mit.edu/andoni/LSH/
The context-base should provide a generic mechanism to make optimized retrieval possible for all Aspects, in any kind of combination. The idea was to narrow the problem scope to make searching more realistically applicable in real time applications. The final method (see getEntitiesByContext in 3.2.5) lets the user specify ranges for each constraining Aspect.

Due to the implementation's ability to compare entries context information by their specified metric (specified by the Aspects AspectComparator), the assumption can be made that all entries of a certain Dimension of an Aspect are stored in an order.

The sum of all Aspects Dimensions, by which the Entities are filtered specifies the $n$. Suppose a $n$-dimensional retrieval is triggered. The idea now is to compare the Aspects meta data in the context-base (how many entries, how many dimensions, how many possible values in the domain etc.) and the query properties (how much of the domain in percentage is filtered, what type of filter is it?) to decide for the optimal first cut Aspect filter, probably cutting off the most undesired context data for further analysis. This can be compared to the mechanisms used by RDBMS’s for query optimization. The current implementation assumes that a uniform distribution for each Aspects Dimension is given. This allows for predictions regarding the expected amount of retrieved entries for an applied filter.
The pseudo-code algorithm used for retrieval is presented as follows:

```python
getEntitiesByContext(filters, subEntityFilter, refPoint):
    optFilter = chooseOptimalFilter(filters)
    contextValues = retrieveContextValues(optFilter)
    results = []

    for cv in contextValues:
        entities = getHolders(cv, subEntityFilter)
        for entity in entities:
            if notInResults(entity):
                if isValid(entity, filters - optFilter):
                    results.add(entity)

    return sort(results)
```

The filters in the signature are the Aspects with the n-dimensions by which Entities are filtered. The reference point is a ContextValue for one of the filters Aspects. The matching Entities will be ordered with respect to the distance to that point, applying the Aspect’s AspectComparator implementation. The chooseOptimalFilter method is comparing the filters depending on the context-base data and its structure. It simply looks at the percentage of the domain space, for all dimensions of a certain aspect and relates it to the amount of context entries of this aspect in the context-base. The function is:

$$
utilityFilter(p, a) = p \times a
$$

where $p \in \{0, \ldots, 1\}\cap \mathbb{R}$ is the percentage of context values that are expected to be retrieved (considering a uniform distribution). $a$ is the amount of concrete context values in the context-base.

**Complexity**

The previously presented algorithm shows that the problem that has to be solved is slightly different to k-NN. In k-NN the entries that are filtered by are the same as the entries that are requested. In the case of MediaSense the entries that are filtered by (ContextValues) are not the entries that are requested (Entities). The entries requested have ContextValues attached to them. This context could be shared between different Entities. The consequence is that in general the problem is computationally more expensive: For each ContextValue, all Entities have to be checked for validity regarding the filters, by matching their further attached ContextValues. This makes a worst case of all Entities being attached to all ContextValues.

In terms of querying complexity, it has to be considered:

1. the amount $cv$ of ContextValues, retrieved after the first cut.
2. the sum of Dimensions $n$ for the applied Aspects filters.
3. The amount of Entities $e_{cv}$, holding the respective InformationSource for the ContextValues $cv$. 

Table 4.1: An example for a fictive Entities - Router and it’s context.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Amount Entries</th>
<th>Domain</th>
<th>Domain Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>1000</td>
<td>{true, false}</td>
<td>2</td>
</tr>
<tr>
<td>Throughput</td>
<td>1000</td>
<td>{0,...,100.000}kbit/s</td>
<td>inf</td>
</tr>
<tr>
<td>Location</td>
<td>1000</td>
<td>{-180,...,180}</td>
<td>inf</td>
</tr>
</tbody>
</table>

A query then involves \(query_{range}(cv, n, e[])\) checks for Entity validity, with

\[
query_{range}(cv, n, e[]) = \sum_{e_cv \in e[]} cv * e_cv * n
\]

Example

The following depicts an example, where Entities of type Router are filtered by their context. See table 4.1 for the setup. Filtering with respect to all three aspects is defined by following setup:

- Throughput \(> 90.000 \text{ kbit/s}\)
- Activity = true
- Location: \((-120 < \text{latitude} < -100) \& (0 < \text{longitude} < 20)\)

So the active routers in a certain delta, carrying out a very high load are requested. The retrieval specifies 4 criteria for a 4 dimensional search. The task now is to compute \(p\) and \(a\) respectively for each Aspect. For each Aspect it is important to compute the absolute domain range and the range for each filter first.

\[
\begin{align*}
\text{range}_T &= \text{dist}(0, 100.000) = 100.000 \\
\text{range}_F &= \text{dist}(90.000, 100.000) = 10.000 \\
\text{range}_{L_{lat}} &= \text{range}_{L_{lon}} = \text{dist}(-180, 180) = 360 \\
\text{range}_{F_{lat}} &= \text{dist}(-120, -100) = 20 \\
\text{range}_{F_{lon}} &= \text{dist}(0, 20) = 20
\end{align*}
\]

Table 4.1 already tells us \(\text{range}_A = 2\) and \(\text{range}_{FA} = 1\) due to the discrete property of the Aspect. To compute \(p\) we do:

\[
\begin{align*}
p_A &= \frac{\text{range}_{FA}}{\text{range}_A} = \frac{1}{2} = 0.5 \\
p_T &= \frac{\text{range}_F}{\text{range}_T} = \frac{10.000}{100.000} = 0.1 \\
p_L &= \frac{\text{range}_{lat} \ast \text{range}_{lon}}{(\text{range}_{lat} \ast \text{range}_{lon})} = \frac{400}{129600} = 0.0031
\end{align*}
\]

The amount of entries in the context-base is the same for all aspects, 1000. Therefore the computations are:

\[
\begin{align*}
\text{utilityFilter}(T) &= p_T \ast a_T = 0.1 \ast 1000 = 100 \\
\text{utilityFilter}(A) &= p_A \ast a_T = 0.5 \ast 1000 = 500 \\
\text{utilityFilter}(L) &= p_L \ast a_T = 0.0031 \ast 1000 \approx 3
\end{align*}
\]

So it can be expected that filter \(T\) reduces the set to 100 potential candidates, filter \(A\) to 500 and filter \(L\) to 3.

This result makes it simple to pick a first filter. \(L\) filters significantly more
entries than $T$ or $A$. This is just an example of how the algorithm decides upon the filters and entries in the context-base.

It should be mentioned that routers, not supporting all three Aspects, are neglected as a possible result. Applying the wildcard pattern is simply done by specifying a filter without criteria.

Since routers in general are not placed uniformly distributed over the planet, a uniform distribution could give bad approximations. It should be considered for future work to let modeler define distributions for Aspects, Representations or as well specific Entities like Routers.
5 Discussion

In Chapter 5 the overall importance and applicability of the presented work, with respect to the problem statement and context modeling in general, will be discussed. For this reason the chapter is split in two: CII-model related discussion and implementation related discussion. Where matters are overlapping, this is stated within the respective sections.

5.1 CII-Model

5.1.1 General

Firstly, the question that should be raised is, why a new context model at all? There are many context models around, applied to certain problem domains. The proposed model is a combination of former efforts, ensuring a simple concept structure and emphasizing it’s usage in distributed sensor networks.

A model like CII had not been found by the author, even though it’s main concepts where used in different approaches before. The models that had been found by the author where in general either:

- domain specific and therefore of no generic use or
- involve the specification of an entire context distribution architecture.

This already tells that the decision for a model and for the final implementation were tightly coupled by protocols or technology for most of the literature. This work tries to avoid making assumptions on how the context-base as a service is enabled, rather stressing it’s core features.

But what are these main concepts that combined bring a higher value to the model? Here is the enumeration of the key concepts:

Interlingua The concept of inter-lingua can be applied on many different levels of abstraction. For instance for heterogenous network integration via gateways, language independent data processing by XML, or units for measurement of sensor data like temperature that are of different type (e.g. Celsius, Fahrenheit). The CII-model inherits it’s second 'I' from the inter-lingua principle, because it ”integrates” context from different sources and diverging formats into one standard. The model requires a consensus about the standard, otherwise it can not be applied. This forcing makes the integration of context much easier and as well more performant. In the CII-model, each format is expressed as a Representation. It’s specifier has to ensure that translation from and to the standard Representation is supported.
Aspects, as defined in the Aspect-Scale-Context model in Korbinian et al [8], provide a simple means to specify/understand the context semantics. They serve as meta descriptors for the concrete context data which is exchanged.

**Ontology** Modeling the meta-model, using an upper-ontology, ensures extendability and adaptability to any many sensor network problem domains.

**Social Networks** are enabled by the correlations between Entities in the context-base. Entities and its sub-concepts build a graph that thereby supports traversing from very close context to context which is further away and therefore probably of less significance. This structural network can be used to semantically interpret context, assuming the structure of the social network had been exploited (see context definitions in chapter two). Since context is very subjective, these graphs make it up to the user to weight importance of attached context for a concrete situation. When comparing with the semantic web approach and its RDF triples, the proposed work still has to add minor extensions to make it as expressive.

A sensor network model can benefit from these features. Sensor data that covers the same Aspects (e.g. GPS data) might be expressed in different formats or units and therefore requires a means for integration; an interlingua. To specify sensor context by its type, a concept for appropriate management is required; the Aspect pattern could serve for that. The supported sensor collection could easily be extended by the specification of new InformationSource sub-concepts. For retrieval and interpretation of context for users, groups or places holding sensors, the social network could serve. The abstract definition of the social network allows an open interpretation on how relations are to be seen. The most common structure for these graphs is the triple concept: subject-predicate-object (S,P,O), where the subject S and object O are two entities, that are related to each other by the predicate P. At the moment arcs between entities have no predicates and only form hierarchies (see 2.3.5). The semantic extension is one thing that should be considered for future work. This triple pattern is not yet semantically expressible, but it would not require a major model change to add the required concepts.

### 5.1.2 Flexibility

The CII-model allows the definition of domain specific models of any kind. Appendix A, Use Cases, shows that a traffic information system, a ad-hoc tracking system and a broker for messengers can all be modeled and integrated into one common data structure.

The flexibility could become a problem if the standard Representations chosen for an Aspect are not sufficiently expressive for a new domain. A simple abstract example might be an existing Dimension with values in \( \mathbb{N} \cap \{1, \ldots, 10\} \), where the granularity should be extended to 11 values. If this is recognized when the model is already in use, the problem at hand is that even mapping to other Representations can not possibly bring along a richer expressiveness. To solve this, either a similar Aspect with a similar, but more expressive, standard Representation has to be defined, or a rather
complex standard reset has to be applied. The latter solution is the cleaner one, since it does not violate the integration of different Representations.

5.2 Approach

5.2.1 General

The model itself is just an abstract specification. A validation of it’s applicability in terms of feasibility, scalability and performance has to be conducted. Without a concrete implementation it resides a theory. The approach for the specification of the model and the derivation of a suiting framework for the implementation was use-case driven with respect to two matters:

1. the technological applicability of the approach, suiting the time frame of a master thesis. But at least likewise important

2. the applicability of the model to solve problems in a comprehensive, traceable manner.

So there had been an all time perspective swift between application centric and user centric development. The user centric viewpoint was much influenced by the three disparate use cases in appendix A. The application centric perspective was influenced by the definition of a clear user interface for the extension and consumption of the model and the research conducted on suiting frameworks to handle the problematic substance with regards to real time applications. Even though performance and scalability tuning was not part of the assignment, this work should contribute in a way that it can be significantly improved with respect to these points.

Fundamental Decision

Due to the Internets decentralized architecture, the semantic web approach did not possibly have a chance to organize data in a more centralized manner. An extension of the existing structures was the only possible downward compatible choice. The standard advertisement did not succeed entirely, due to it’s complex semantic extensions. It can simply not be expected from all private users, hosting a web page, to tag the contents following the W3C recommendations. Anyway, a good argumentation for pursuing the standard is that search engines find in general better results for semantic tagged pages and therefore guide the right prospects there.

For services in sensor networks, the need for precise retrieval upon a global knowledge base under real time condition has to be guaranteed. On the long run, the goal must be to reach such a decentralized knowledge representation with certain and sound retrieval, applying an approach like the semantic web. This thesis does not present a solution to all problems concerning decentralized context storage and distribution, but deals with core problems of the context modeling in sensor networks and proposes an implementation that allows for further decentralization. For the first step the assumption is that mobile users have permanent up-links to the Internet, thereby are able to request knowledge and push context directly from wherever they are into the context-base. Now where cloud computing is talked about a lot, this assumption does not primarily require a distributed context-base in the first place. However, the architecture should be sufficiently open so to
allow the implementation behind the cloud to decentralize transparently to the user. The presented approach will have to be decentralized, due to its Single Point Of Failure (SPOF) and bottleneck properties, when serving for possibly millions of users with billions of sensors.

It would have been possible to use a global context-base, by directly applying the Internet standards. Frameworks like Jena, Sesame or Protege deal with the issues of integrating ontologies into object oriented programming languages or the transparent persistence of their concepts. They as well provide the user with interfaces for the usage of different reasoners upon these models. In the proposed architecture they could reset the DataNucleus platform. But since scalability tests show serious problems for large databases ([22], [23]) it felt right to try another, possibly better scaling approach. The DataNucleus platform allows for ODBMS as a choice for persistence. ODBMS are using an object graph, rather than static tables for the persistence of data. Navigation through this graph is fast for today’s ODBMS. They would therefore suite very well for the social Entity networks and navigation through them. Unfortunately, no functional driver for a multi-user ODBMS from within DataNucleus exists at the moment.

By applying object oriented programming patterns, the powerful reflection mechanism and a flexible persistence framework, it was possible to enhance the static Java class models to an extensible solution for runtime integration of storable concepts.

Why the W3C model does not apply

The vast majority of context modeling approaches within the last 6 years are build upon W3C’s recommendations. The author thinks this is the wrong way to go for this particular project, and tries to motivate the decision against a direct usage of the semantic web standard.

An adaption of the semantic web standard would have caused following issues: 1) the creation of very simple models would require the specifier to have a good knowledge about the semantic web ideas, technologies and standards. It would complicate the application of simple proprietary formats and thereby especially the integration of formats that are intensely used.

As already mentioned, OWL has three different dialects. For the purposes of this thesis it would be crucial to be able to define concepts and sub-concepts, as well as social relationships between entities, allowing for instance transitivity relationships. Therefore, the only possible choice for an extensible model would have been OWL Full. OWL full is not decidable, what means that infinite queries could easily be issued by a user, intruding the whole context-base. A mechanism would be required to a) either check queries for decidability, what is verifiable not possible or b) stop each query after a certain time-frame, disallowing services with long running queries. A priority-mechanism for issued queries would be useful in that case. But no one can guarantee that someone with a high priority and therefore long retrieval times, is not entering a never ending query process.
One consequence of the implementation is that the model is expressed in classes, not as ontological concepts. In general, this has a large impact on reasoning. The advantage of using an object oriented model is that it is the commonly used programming language pattern. Therefore a large amount of tooling exist with respect to data processing, persistence and distribution. Further more it is abstracting the developer from the concrete data representation, leaving him in the well known object oriented environment. This enables rather a jump start into the modeling of context models.

Anyway, compared to ontologies it has some disadvantages, especially with respect to reasoning.

**Disadvantages of the Implementation**

The type of an object is defined by it’s class, and all it’s super classes. A class can be instance of more than one class at the same time. However, an object is statically assigned to these classes; once it is created, their types will remain static. In ontologies, concepts are classified by their attributes. Because attributes can change at runtime for ontologies, their types might change as well. Object attributes don’t change at runtime, in fact an objects state is specified as the set of values of it’s attributes.

Figure 5.2.1 tries to visualize the difference. Where the object oriented specification looks very much like the ontological, they have different semantics. If in an object oriented model all instances of type A are requested, the issuer receives all the instances that explicitly have been assigned to A at initialization time. If A and B stand in no inheritance relation, no objects of type B will be returned. Querying all concepts of type A in an ontology retrieves the exact same amount of instances, as a query for concepts of type B. Querying in ontologies is therefore more powerful, because at runtime there exists no static structure, as it does for classes. Using an object oriented implementation does not mean that the W3C recommended mechanisms are not usable in the model. Context values themselves could for instance be described in OWL syntax and therefore contain more than just simple aspect ⇒ representation ⇒ value pairs. The acquisition of relevant
context values could be conducted and reasoning could be applied.

The CII model in its proposed version is not as expressive as OWL, when it comes to the social network. Only directed unnamed relations between the entities are definable. An example of these could be that an instance of type Person, Peter, has a directed arc to an instance of type Car, HH3-331, which can semantically be interpreted as 'Peter owns a car with signature HH3-331'. The extension of the model to define named predicates would close this semantic gap.

5.2.2 Extendability

The global availability of the model and the runtime integration of new concepts make the model very extensible in the sense that new aspects, Representations, sub-Entities or sub-InformationSources can be added. The reuseability of the model is therefore significant, because all contents is available for everyone with a JVM and access to the global context-base.

What still has to be created is a means to communicate existing structures to developers, so no redundancy and therefore divergence of context and concepts evolves.

Live data and history data are physically located inside the same database. Therefore it is possible to use history tables as well as live context tables in one query. Further research has to be conducted for the possible integration of tables from different data-sources in single queries.

5.2.3 Openness

The model could serve as a public 'context model tool box' where everyone may leave his concepts and use the concepts of the others. By the definition of common Aspects, Representations, InformationSources and Entities, it is simple to build applications that combine context from many context sources. Each format is allowed, from proprietary to XML-schema. So not only the context itself is shared throughout the base, the concepts are shared as well.

5.2.4 Reasoning

Using the DataNucleus access platform, allows for the usage of other DataNucleus products, like the analysis platform. The analysis platform provides the user with means for data analysis and simple access to meta data of the models, created with the access platform. This might solve the biggest issue of the implementation, the lacking build in reasoning functionality. As a matter of fact, there is no current implementation of the analysis platform available. The author admits to have overseen that.

Anyway, by using Aspects and Representations, it is possible to give quantified data, qualified interpretations, depending on the context. An example Aspect temperature with a Representation XML_celsius could represent the data as a node of an XML tree, using the Celsius unit. A representation XML_bathingTemp for water could be created, that defines a domain \( d_{\text{bath}} = \{\text{cold, warm, hot}\} \). It’s respective RepresentationMapper
implementation is responsible for the mapping which could look like the following:

\[ f(x) = \begin{cases} 
  \text{cold}, & x \leq 14 \\
  \text{warm}, & 14 < x < 33 \\
  \text{hot}, & x > 33 
\end{cases} \]

This way, semantic retrieval is possible, depending on the contextual interest (for instance bathing). The disadvantage is anyway, that this can not be defined ad-hoc, but rather in advanced for known purposes and prospects. A human for instance has other demands for a *good bath* than a reptile.

### 5.2.5 Security

Even though security was not stressed by this work, it is important to be discussed anyway. Security does not only concern the usage of the code-base (authentication) and the access rights to the context data (authorization), but as well the rights to use certain concepts that exist globally in the context-base. These concepts could be subclasses of `Entity`, for instance agents that act on behalf of the user.

Where authentication could simply be added by password access or biometric features, authorization is much more tricky. The enrollment for access control lists or capabilities would be very complicated due to the fact that all `ContextValues` can be queried by the retrieval engine, which is externalized to the user. Applying Access Control Lists (ACLs), each database entry would have to be checked after retrieval towards the user rights. Since the system holds possibly millions of entries, it might become a huge administrative work to grant rights correctly to the users.

### 5.2.6 Scalability

Scalability could become a serious problem for the implementation. There is one physical database holding the context. It has to be validated to what extent the usage of this model is scalable regarding general MediaSense requirements and in particular special context-based services that build up on the context-base.

Anyway, there are many ways to scale up the system when user amounts grow. The database tables could be distributed to different nodes. A motivation for that might be to decentralize the load for the context-base server. Distributed transactional databases guarantee ACID (Atomicity, Consistency, Isolation & Durability). However, they require the application of protocols such as 2PC (two phase commit), to ensure consistency.

Further studies could deal with the distribution of database tables for relational databases. JDBC keeps the distribution transparent from the user, so no layers above the access platform layer in the architecture would be effected.

### 5.2.7 Performance

The context-base performance was not emphasized by this research. There is much open space for improvement. Tools for session reuse and query optimization for the DataNucleus access platform exist and their parameters
Figure 5.2.2: Inheritance can improve performance in database schemes.

It is to be expected that sooner or later an object oriented database is either newly coming out or an existing is enhanced, to support transactional behavior for multi-user requirements in a scalable manner. DataNucleus already allows the integration of two ODBMS (db4o, ObjectDB), so switching to one of them would not bring along any code changes. An object oriented database is expected to be much faster, because persisted data is directly present in the business layer of the application; no extra database connections have to be established, opened and closed.

Each change of the DataNucleus access platform with respect to performance, security or reasoning features will be directly inherited by the the CII implementation. In case the object oriented querying language obtains improvement, it is immediately available after an access platform updated.

**Build In query optimization**

The model itself, by its flexible concept extension, does improve performance. Picture 5.2.2 shows the optimization for InformationSources (the same pattern applies for Entities). The figure shows that n InformationSources exist in total in the context-base. These n are the sum of all instantiatable InformationSources $k_i$:

$$n = \sum_{i \in K} k_i$$

Queries can be optimized to only consider $k_i$ candidates; the ones of a specific type, when requested. This in general results in a significant performance gain. All other concepts that are not of the specified type are neglected for the search.
5.2.8 Flexibility

The implementation offers something that does not necessarily have to be part of the model: persistent code. If the developer defines an Entity with attached methods and fields, it’s code becomes stored as well and can be retrieved from any application with access to the context-base. The methods could be very simple field setter/getter. They may contain arbitrary Java, what makes them Turing powerful.

Sub-concepts of Entity and InformationSource can be defined by domain developers, building a conceptual-hierarchy. This might be useful to improve reuseability (and therefore integration) and when it comes to the implementation and retrieval, performance.

So, classes can be uploaded to the global base and queries from each application with access to the context-base can retrieve an instance of specific types. Any method could be started as entirely independent thread. A thread could perform background jobs, cleaning up the code-base history, or run more user oriented tasks. These programs can be referred to as agents, because they autonomously solve tasks in order to pursue someone else’s goal. They could act independently, or cooperate with each other by socket communication, or any other medium exposed by Java. It allows multi-agent communities, running on different devices, reasoning or acting upon the global model. Since these agents are part of the code-base, and therefore potentially first class objects with a stored state, they can reason upon all context within the context-base, including their own. That makes them self-aware. Self-awareness might be important where the state of an agent has an impact on the providence of a service. For example an agent on a mobile phone understands that it’s capabilities for the execution of a task are insufficient (battery, computational power) and therefore a) delegates the task to a cooperative agent by a socket request like in cloud computing or b) returns an error message to the user, telling that the current service is not executable under the environmental conditions. Figure 5.2.3 shows the four step process that leads to the overall usability of an agent in the model. Since agents are a part of the social Entity-community, they can be retrieved like all other storable objects.

Further more, there are no restrictions to the logic’s that are used for retrieving or reasoning upon the model. The agents could make decisions on temporal activity (‘as long as X do Y’) in temporal or even modal logic (‘since agent X thinks Y because of it’s state Z, I do A’). So the agents are not only self-aware, but as well other-aware, meaning that they know what others think (by their storable states). However, the activity of reasoning will not be expressible in a descriptive manner (what is requested) but rather in a procedural manner (how it is requested).

5.2.9 Searchability

The Java Context Framework was an initiative to model arbitrary context, comparable to this thesis approach. However, it does not have any means for context retrieval at all and uses simple key-value pairs. Context could only be accessed programmatically.
For this thesis it was crucial to enable descriptive queries for a flexible and simple reuse of concepts. One of the main problematics was to find an adequate mechanism to integrate data of all formats, making it retrievable by only one common interface and language. DataNucleus gave the answer by offering support to three languages: SQL, JDOQL, JPQL. Where SQL is only applicable for relational databases, and therefore does not keep the generality of the approach, JDOQL and JPQL are object querying languages that are translated by DataNucleus to whatever supported data-source. The choice for JDOQL as querying language could be discussed. It is simple to use and allows object oriented principles like object id’s and inheritance. But JDOQL is no standard. There are no books on the market that explain the usage and it is not widely applied.

The ontology querying language SPARQL is more commonly used and subject of much discussion in industry research. But it would have forced the author to use the semantic web approach. Due to the time restrictions and formerly discussed drawbacks, he decided to stick to DataNucleus in combination with JDOQL for the prototype.

By assigning new Representations to Aspects with qualitative measures, mapping them to the quantitative standard Representation, semantics like "good", "bad", "mediocre" are expressible. Semantic context searching might be more applicable by the entity context retrieval method for n-dimensional spaces.
Logical Search Step

Due to the fact that a relational database is the concrete structure for storage, the complexity for retrieval is well explored and depends very much on the optimization of the relational algebra for SQL calls by the RDBMS. In Elmasri et al [32] the author explains precisely how to measure query complexity and deals with the optimization of queries.

N-Dimensional Context Retrieval

The n-dimensional context retrieval allows a straightforward filtering of Entities by the application of ranges or value filters. It is not as powerful as a k-NN implementation, but a lot faster.

When referring to the complexity properties of Entity context retrieval, it may in theory not be sufficiently fast. But in most of the cases, it will be very fast, especially when the filters are expected to cut a huge amount of candidates for a retrieval. It may not be forgotten that k-NN is just too problematic in terms of computability and that tricks have to be applied to avoid the existing complexity problem in the general case.

k-NN is a NP-complete problem. It had been shown that NP-complete problems can all be reduced to each other and can not be solved within polynomial time complexity. This matter makes things very hard for real time applications.

The MediaSense project deals with highly dynamic environments and therefore requires a context-base with acceptable response times for services dealing with real time issues. For the time being, it should be acceptable to search in n-dimensional ranges for neighbors, due to the complexity problems for k-NN retrieval. This way, the system avoids complicated data structures, which maintenance would take a lot of extra resources. In future work this problem could be tackled, if it shows that certain services are not feasible by this approach.

However, this restriction disables the opportunity to pinpoint an entity and request the k-NN around. The caller has to guess the area in which the interesting entities remain.

That results are returned ordered with respect to an at compile time unknown Aspect, allows for the developer to specify any orders with arbitrary metrics. That makes the context-based retrieval very powerful. User could define metrics and the context-base uses these for ordering comparing internally.
6 Conclusion

A context modeling approach combining object oriented design, ontologies and markup scheme languages have been presented. The model is capable of representing arbitrary context data. This context data can at runtime be attached and detached from entities in the virtual community. These entities could either be very abstract things like groups or very concrete things like persons or cars.

One of the key features of the model is the integration of arbitrary context representations into one common structure. That allows for retrieval and reasoning upon the context, as if it was following one representation.

An API was proposed and an architecture including a prototype were presented and discussed. The prototype is capable of context persistence and retrieval.

The author proposed an approach to tackle the problem of context-storage. He has knowingly chosen a different approach than the commonly applied one for the semantic web, trying to overcome it’s scalability issues.

To model in objects is very simple for developers to adapt to, in contrast to the learning of ontology specification languages and their very high level of abstraction. However, it has to be validated in future work if the context model with it’s current implementation suffices the targeted service space for mobile services to a satisfying extend.

If there is a need for semantic inference support, the prototypes applicability has to be discussed and evaluated. An RDF based access platform would be recommended in that case. But since these scenarios are not drawn yet and the possibilities of the implementation not entirely explored, the author can not draw a conclusion on that at the time being.

Section Future Work discusses into which directions further studies and implementation could go.

The consequences of a global CII adaption would be that context data is integrable using arbitrary proprietary formats between any two applications. The implementation is based on Java technology, which already today is used as mobile platform by the majority of devices. Especially in mobile environments it would be interesting to see how the requirements for real context-aware applications are covered by the model, when more advanced domains are to be considered.
6.1 Future Work

The overall adaption of a common context ontology and a scalable implementation will be the key drivers for interoperability. Context-aware services would not only be based on simple locational data, but as well on the anticipation of user behaviors. User surveillance by sophisticated agents, anticipating the users goals and desires could be a way to get there. Until then, the CII prototype is applicable only in a testing environment; many obstacles still have to be overcome. The one completely left out of the discussion in this thesis, but maybe the most significant, is the one of privacy and security in general. The global sharing of context means the global sharing of local private data, which in many countries generally violates an individuals privacy. The global sharing over country borders therefore requires investigation and a very general policy structure, as well as communication protocol for trustworthy authorization and authentication deployment. This seems like a very complex enterprise, and complex enterprises are failure prone themselves. That makes global context sharing a very challenging area, not only with respect to modeling and the applied technology. However, this work should show that the technical possibility for context integration exists.

Reasoning upon the model should be further explored to see what opportunities are present for automated context based applications. The author could imagine to work as a researcher/developer/architect with applications and middleware issues that are highly dependent on or serve as a means for context-awareness.

Further investigation of semantic web tools for context integration did not fit into the time-frame of this thesis. It would be interesting to investigate on the edge Internet technology for data integration like Mashups into the discussion.

Most importantly, performance and feasibility tests have to be conducted. The context-base would have to be applied to context-aware applications in real scenarios to improve the implementation by facing concrete problems.

6.1.1 CII on top of DCXP

The perspective taken in this thesis when considering DCXP is that the context-base runs as a node in the overlay. Each stakeholder has to push context to and pull context from this central unit. It might be interesting to integrate the CII model into DCXP to make context distributed; re-setting or complementing DCXP’s distributed hash table.

Reviewing DCXP, there are five primitives for a user agent [15]:

- Register
- Resolve
- Get
- Subscribe
- Notify
The CII-model could be build into the protocol, by triggering following activities when a primitive is issued:

**Register:** Add an `InformationSource` to the context-base.

**Get:** Retrieve value from context-base.

**Notify:** Update a certain `ContextValue` of an `InformationSource` to the context-base. This involves the archiving of the former valid value.

This could be seen as an alternative or a reset for the DHT. As an alternative it could lead to faster replies due to the fact that queries are issued to both, the DHT and the context-base, while only the one that replies fastest is processed by the user agent.

The database could in future work itself be distributed over many nodes. DCXP has to ensure that the issued, context-base involving, commands reach a node that actually has access to the context-base interface. How context is physically retrieved from that node on is then again the context-base's problem. The prototype would deal with this via a remote classpath setup and a JDBC connection to the database.

### 6.1.2 Implementation

One of the most important aspects of the future work is the improvement of the implementation with respect to performance and scalability. The existing implementation could be seen as a prototype, which already delivers the basic functionalities Some of the most significant todo’s are listed here:

**Improving the n-Dimensional Context Retrieval** At the moment, the n-dimensional context retrieval is not implemented to the full satisfaction of the author. It should be for instance possible to filter for more than one range of a `Dimension` of an `Aspect` at a time. Further more, an interface for user defined distributions of context values in their domain should be added. That way, the first cut principle would produce more reliable predictions about the amount of filtered entries for a criteria.

Even though the search capabilities are narrowed to n-dimensional ranges, it should be possible to quantify results to an amount k of nearest neighbors with respect to the entries that match a query. That would reduce traffic to a minimum, so that the amount of possible entries are in the range of \( \{0, \ldots, k\} \). It would not affect the performance of queries, because the method would simply filter the final result to only contain k entries. A simple extension of the methods signature to a user parameter k would allow that.

**Decentralizing the Classpath Dependency** One important extension of the model is to unfasten the context-aware application from a central code-base that shares the global concepts like presented in figure 3.4.8. Two approaches are possible here. One is to split the classpath up into a backbone network of concepts like in figure 6.1.1. Each context-aware application could have it’s own local classpath assigned. A mechanism for node synchronization would be required. Another approach could be to store classes in a database as
well. It would have to be investigated how automated class loading at runtime could be enabled by writing an own class-loader for java. The first approach is probably simpler to realize, where the second would be far more elegant.

**Distributing the Database** There is one physical database at the time being, storing all data. It is accessible by the access platform, by applying JDBC connections. Extending the schema for distributed databases would be rather simple, since much of the context is already split up into a growing amount of tables at runtime. For postgresQL, a tool of choice might be Slony-I [33], which in version 2.0 should be able to deal with distributed databases. A good distribution scheme has to be found.

Figure 6.1.1: An approach to decentralize the global concept base.
References


[10] “Datamucleus project home, last visited 04/09.”


[12] “Sesame project home, last visited 04/09.”

    http://researchdatabase.miun.se/converis/project/402;jsessionid=4b66393f1a7610a4b4225368d95a.
References


[26] “Objectivity/db solutions news page, last visited 04/09.”


[28] “Hibernate project home, last visited 04/09.”
[29] “ibatis project home, last visited 04/09.”


[31] “db4o project home, last visited 04/09.”
   http://www.db4o.com/.


[33] “Slony project home, last visited 04/09.”
   http://slony.info/.
A Use Cases

In this chapter three use cases for context-aware applications are presented. The author tried to find realistic scenarios, addressing entirely different issues. The focus lies here on a) the creation of a domain structure and the steps for the developer to take for b) the integration into the context-base and c) the context retrieval. The purpose of this is to show the applicability of the CII-model and its implementation.

The use cases are presented in a descriptive manner. It follows a more technical description of the processing involved into the problem solving. Further more, a domain model is recommended for each case.

It can be assumed that for the pushing and pulling of context via subscriptions, DCXP in combination with Ericsson’s IMS is used. That would allow a simple user interaction via mobile phones, capable of running java. The Use-case one, the Traffic Congestion Information System is far more extensively exposed than the other two. Some of the Java classes and JDO meta-descriptors are depicted. How model extensions are shared and how context is becoming integrated is discussed as well.

The solutions are just examples of how the CII-model could be applied. Time and context freshness are not considered as an issue. The cases are simplified. It is up to the application developer to find more suitable implementations; for instance more efficient ones.

The development for use-case one requires the developer to finish the tutorial in appendix A. It is a step by step guide describing how to setup the environment and start the development. The traffic example had been developed and resides on the MediaSense Subversion server for project wide access.

A.1 Traffic Information System

Suppose, the user is driving on a highway on rush hour into town. The car he drives has an embedded traffic information system, which dynamically reacts on traffic changes, recommending alternative routes in case of traffic congestion. Each car holds a location, a velocity and an orientation sensor. It frequently updates it’s respective sensor context via an up-link (e.g. GSM from mobile phone) to the context-base. A feedback for traffic related incidents in the near or a recommendation for route change are possible context-based services that could save the users life or speed up his ride.

Figure A.1.1 shows the three stakeholders. There is the car driver on the left
side of the figure, consuming the service. The traffic information system is
the service provider in the center of action. Lastly there is the context-base,
allowing context dependant reasoning.

Figure A.1.1: The traffic information system. Driver and TrafficInformationSystem can both be seen as service consumers, just that the driver is consuming the services of TrafficInformationSystem and the TrafficInformationSystem the services of the ContextStore.

A.1.1 Traffic Service

Following this idea, the service could be implemented to either reason locally
and therefore serve for only one car or reason globally, so to enable the service
for all interested drivers. Figure A.1.2 visualizes these two approaches. On
the left side the simpler architecture with it’s local reasoner can be seen. The
car itself contains a requesting unit that filters the context of interest
from the context-base and delegates it to the local reasoner. Depending
on the contents, this reasoner reports suspicious behavior to the warning
system, which signals the driver.
On the right side, one intermediate unit is taking over the polling and reasoning. Only in case of suspicious activities, a warning (or recommendation) is send to the car, which receives it via it’s down-link. The repeating process could be described like in A.1.3.

The latter approach is probably the preferred, since it is rather scalable.

Due to it’s centralized reasoner, many cars can be informed about problems after one global reasoning run. In the local reasoner example each car reasons itself and by that increases the general data traffic immensely.

In this example the reasoners work is neglected.

A.1.2 Domain Model

The context domain model which has to be defined by the service developer could look like in figure A.1.4.

Starting out from the top, a sub-concept of Entity, Car is specified. All concrete cars in the model will carry this type. Each car holds three sensors; location, orientation and velocity. Since all these three concepts address physical sensors, a new sub-concept of InformationSource by the name Sensor is defined. Each of these sensors now covers a certain measurable Aspect of the world. A sensor of Aspect location for instance does by some means sense and represent locational data. The class LocationComparator deals with the mapping between different location representing units and formats. Latitude and longitude (elevation is neglected for simplicity) are considered as the Dimensions that define a location. The KMLPoint serves here as the locational standard Representation. The KmlMapper has to be implemented to ensure extendability for the inter-lingua pattern. The
Figure A.1.3: The abstract repetitive process that the traffic information system controls.
definition of the concepts with respect to orientation and velocity sensors are similar to the one for location.

![Diagram of context model](image)

Figure A.1.4: *A possible model extension for the traffic information system scenario. Cars are carrying sensors of three different types. Each of these types has attached meta data about it’s Aspects Dimensions and Representation.*

**Model Implementation**

It follows an enumeration of concepts for the traffic congestion service.

**The Car** A car and it’s context are to be stored in the context-base. The domain model must define a) a Java class `Car`, extending `Entity` (Car.java) and b) a JDO meta-descriptor for this class (Car.jdo). In case cars don’t hold any other fields to store or methods than an Entity, the simple code snippet in figure A.1.5 shows the implementation. Figure A.1.6 shows the meta-descriptor.

**The Sensors** For the sensors everything applies that applies for the cars. As an example the `GpsSensor` implementation can be seen in code snippet A.1.7. Notice: The `GpsSensor` is not inheriting directly from `Information-...
/**
 * A sub-class [sub-concept] of class [concept] Entity.
 * @author Felix Dobslaw
 */

public class Car extends Entity {

    // the empty standard constructor [existence mandatory for JDO persistence]
    protected Car() {
    }

    /**
     * The minimal constructor for a sub-class of of Entity.
     * An id is necessary for the identification in the context-base.
     */
    public Car(String id) {
        super(id);
    }
}

Figure A.1.5: The sample code for a sub-class Car of Entity, ready for the integration into the context-base.

<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC
"//Sun Microsystems, Inc./DTD Java Data Objects Metadata 2.0/EN",
"http://java.sun.com/dtd/jdo_2_0.dtd">
<jdo>
    <package name="se.num.mediasense.context.model.entity">
        <class name="Car">
            <inheritance strategy="new-table"/>
        </class>
    </package>
</jdo>

Figure A.1.6: The minimal meta-descriptor for the Car class, which should be made storable.
Source, but from Sensor, which itself is inheriting InformationSource. This pattern could be applied for arbitrary heredity depth.

```java
/**
 * A sub-class (sub-concept) of class (concept) Sensor, which itself is a
 * sub-class of class InformationSource.
 * (A well maintained hierarchy increases performance for retrieval)
 * @author Felix Dobslaw
 */
public class GpsSensor extends Sensor {

    // the empty standard constructor [existence mandatory for JDO persistence]
    protected GpsSensor() {}

    /**
     * The minimal constructor for a sub-class of InformationSource.
     * An id is necessary for the identification in the context-base.
     * The representationId links to the right context-base quadrant.
     * @param id
     * @param representationId
     */
    public GpsSensor(String isId, String representationId) {
        super(isId, representationId);
    }
}
```

Figure A.1.7: Sample code for a sub-class GpsSensor of Sensor.

```xml
<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 2.0//EN"
'http://java.sun.com/dtd/jdo_2_0.dtd'>
<jdo>
    <package name="se.nium.mediasense.context.model.is">
        <class name="GpsSensor">
            <inheritance strategy="new-table"/>
        </class>
    </package>
</jdo>
```

Figure A.1.8: The minimal meta-descriptor for the GpsSensor class.

The IAspectComparator principle is exemplified by the Location-Comparator. It implements the interface IAspectComparator which can be seen in C.1.1. Figure A.1.9 shows the comparator code.

The IRepresentationMapper The KmlMapper for Representation Kml_Point for Aspect location was chosen as an example. It implements the interface IRepresentationMapper which can be seen in C.1.1. Figure A.1.10 depicts the implementation code.

Summarizing, the presented code examples should suffice the readers requirements for the creation of CII-model extensions. Now, where the extensions exist, the next step is to make these concepts available to the context-base.
/**
 * The comparator for the aspect 'Location'.
 * @author Felix Dobslaw
 */

public class LocationComparator implements IAspectComparator {

    public static final String LON = 'longitude';
    public static final String LAT = 'latitude';

    /**
     * compare 2 values of type 'longitude' or 'latitude' with each other.
     * for the definition of output values, @see java.lang.Comparable
     */
    public int compareTo(Object rawValue1, Object rawValue2, String dimension) throws Exception {
        int flag = .100;
        if (dimension.equals(LON)) {
            flag = (convertToDouble(rawValue1)).compareTo((convertToDouble(rawValue2)));
        } else if (dimension.equals(LAT)) {
            flag = (convertToDouble(rawValue1)).compareTo((convertToDouble(rawValue2)));
        }
        return flag;
    }

    /**
     * get the distance for 2 locations, relative to either
     * 'longitude' or 'latitude'.
     */
    public BigDecimal getDistance(Object rawValue1, Object rawValue2, String dimension) throws Exception {
        BigDecimal distance = new BigDecimal(Math.sqrt(
            Math.pow((convertToDouble(rawValue1)), (convertToDouble(rawValue2)), 2));
        return distance;
    }

    /**
     * get the distance for 2 locations, considering all
     * dimensions ('longitude' & 'latitude').
     * In this example euclidean.
     */
    public BigDecimal getDistance(HashMap<String, Object> value1, HashMap<String, Object> value2) throws Exception {
        Double value1Long = convertToDouble(value1.get(LON));
        Double value1Lat = convertToDouble(value1.get(LAT));
        Double value2Long = convertToDouble(value2.get(LON));
        Double value2Lat = convertToDouble(value2.get(LAT));
        //euclidean: SQRT ((LON1 - LON2)^2 + (LAT1 - LAT2)^2)
        return new BigDecimal(Math.sqrt(Math.pow(
            (convertToDouble(value1Long)) - (convertToDouble(value2Long)), 2) + Math.pow((convertToDouble(value1Lat)) - (convertToDouble(value2Lat)), 2)));
    }

    // just a help function
    private Double convertToDouble(Object rawValue) {}
/**
 * The mapper for the representation "KML_POINT" for the aspect "Location".
 * @author Felix Dobslaw
 */

public class KmlMapper implements IRepresentatonMapper {
    public static final String KML_POINT = "KML_POINT";
    public static final String LAT = "latitude";
    public static final String LONG = "longitude";
    private DocumentBuilder parser; // a java DOM parser
    private String representationId;

    /**
     * The minimal constructor for each mapper, containing the representation Id.
     */
    public KmlMapper(String representationId) throws ParserConfigurationException {
        parser = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        this.representationId = representationId;
    }

    /**
     * creates a KmlContextValue from the raw values of dimension "latitude" and "longitude".
     * example: <Point><coordinates>122.0522035425693,37.4228990140251</coordinates></Point>
     */
    public Object createContextValue(HashMap rawValues) throws Exception {
        Object longitude = rawValues.get(LONG);
        Object latitude = rawValues.get(LAT);
        StringBuilder builder = new StringBuilder("<Point><coordinates>" + longitude);
        builder.append(',');
        builder.append(latitude);
        builder.append("</coordinates></Point>");
        return builder.toString();
    }

    /**
     * given a concrete value, syntactically specified by the representation "Kml",
     * this method extracts the dimensions 'longitude' and 'latitude' into a hashmap.
     */
    public HashMap extractRawValues(Object value) throws Exception {
        HashMap<String, Object> rawValues = new HashMap<String, Object>();
        Document doc = parser.parse(value);
        Node pointNode = doc.getFirstChild();
        Node coordinateNode = pointNode.getFirstChild();
        String allValues = coordinateNode.getTextContent();
        String[] entries = allValues.split(',');
        Double longitude = Double.parseDouble(entries[0]);
        Double latitude = Double.parseDouble(entries[1]);
        rawValues.put(LONG, longitude);
        rawValues.put(LAT, latitude);
        return rawValues;
    }

    /**
     * This, and all the other mapping methods, are trivial, because "Kml" is
     * the standard representation type of aspect "Location".
     */
    public Object mapFromAspectToStandard(Object value) throws Exception {
        return value;
    }

    public Object mapFromAspectToStandardRawValue(Object rawValue, String dimension)[]
    public Object mapToAspectFromStandardObject value) throws Exception {
    public Object mapToAspectFromStandardRawValue(Object rawValue, String dimension)[]
    public String getRepresentationId() {}
Context Integration

For the context-base integration of a domain model, the interface IContextSpecification has to be applied. Figure A.1.11 carries on with the example, showing how the concepts Car, Sensor, GPSSensor, LocationComparator and KmlMapper along with a specification of the Aspect location and the Representation KmlPoint become uploaded to the context-base. When this code has successfully been executed, the context-base contains the modeled extensions and is ready for related context creation and retrieval.

```java
IContextSpecification spec = new ContextSpecification();

// upload the LocationComparator class
byte[] comparator = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/comparator/LocationComparator.java");
spec.uploadClassToContextBase("se.mlun.mediaseNSE.context.model.comparator.LocationComparator", comparator);

// upload the LocationMapper class
byte[] mapper = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/mapper/KmlMapper.java");
spec.uploadClassToContextBase("se.mlun.mediaseNSE.context.model.mapper.KmlMapper", mapper);

// add the Aspect Location
Dimension longitude = new Dimension("longitude", "R INTERSEC [-180,..180]", true, 0, "java.lang.Double", "DOUBLE");
Dimension latitude = new Dimension("latitude", "R INTERSEC [-180,..180]", true, 0, "java.lang.Double", "DOUBLE");
Dimension[] dimensions = {longitude, latitude};

spec.addAspect(new Aspect("Location", dimensions, "KML_Point", 
"se.mlun.mediaseNSE.context.model.comparator.LocationComparator"));

// add the Representation KML for Aspect Location
String col = "The KML location representation in XML format. It is simple and used by many applications:" +
" generally google apps (they invented the format)";
String example = 
"<Point><coordinates>-122.883053425669, 37.4228990140251, 0</coordinates></Point>";
String format = "http://www.opengis.net/kml/2.2/"; String type = "KML-Schema"
Description description = new Description("KML_Point", col, type, example, 255); description.setDefinitionFormat();

spec.addRepresentation(new Representation("KML_Point", 
"Location", description, "se.mlun.mediaseNSE.context.model.mapper.KmlMapper"));

// upload the Car class
byte[] carJava = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/entity/Car.java");
byte[] carJdo = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/entity/Car.jdo");
spec.uploadClassToContextBase("se.mlun.mediaseNSE.context.model.entity.Car", carJava, carJdo);

// upload the Sensor class
byte[] sensorJava = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/is/Sensor.java");
byte[] sensorJdo = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/is/Sensor.jdo");
spec.uploadClassToContextBase("se.mlun.mediaseNSE.context.model.is.Sensor", sensorJava, sensorJdo);

// upload the GpsSensor class
sensorJava = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/is/GpsSensor.java");
sensorJdo = MetaModelHelper.getBytesForFile("upload/se/mlun/mediaseNSE/context/model/is/GpsSensor.jdo");
spec.uploadClassToContextBase("se.mlun.mediaseNSE.context.model.is.GpsSensor", sensorJava, sensorJdo);
```

Figure A.1.11: A demonstration of how a domain model can be uploaded to the context-base for global integration.
Context Creation

Now that the foundation for context storage exists, a basic description will be given for how the `IContextConsumption` interface can be used to add context data. The code in A.1.12 adds 100 cars to the context-base with randomly picked context. The context combines the three aspects `location`, `orientation` and `velocity`.

```java
IContextConsumption cons = new ContextConsumption();

// add 100 cars to the map with randomly picked context information.
Car car;
GpsSensor gps;
OrientationSensor os;
VelocitySensor vs;
String velocity;
String orientation;
String location;

for (int i = 1; i < 100; i++) {
    car = new Car("car_" + i);
    cons.addEntity(car);
    gps = new GpsSensor("gps_" + i, "KML_Point");
    vs = new VelocitySensor("vel_" + i, "KML_Proprietary");
    os = new OrientationSensor("orientation_" + i, "Degree_Proprietary");
    cons.addInformationSource(gps);
    cons.addInformationSource(vs);
    cons.addInformationSource(os);
    cons.attachInformationSourceToEntity(gps.getId(), car.getId());
    cons.attachInformationSourceToEntity(vs.getId(), car.getId());
    cons.attachInformationSourceToEntity(os.getId(), car.getId());
    location = decideRandomLocation();
    velocity = decideRandomVelocity();
    orientation = decideRandomOrientation();
    cons.updateContextValue(os.getId(), orientation, Calendar.getInstance(), getTimeInMillis());
    cons.updateContextValue(vs.getId(), velocity, Calendar.getInstance(), getTimeInMillis());
    cons.updateContextValue(gps.getId(), location, Calendar.getInstance(), getTimeInMillis());
}
```

Figure A.1.12: A demonstration of how concrete instances of the traffic model are added to the context-base, so they and their context can be retrieved.

```java
object[] objects = cons.requestContextInformation("SELECT FROM se.run.mediasense.context.model.Entity WHERE " +
    "this.id.matches(".*-\") & this.attachInformationSources.contains(l) & " +
    "toString() == "KML_Point" VARIABLES se.run.mediasense.context.model.InformationSource i");
```

Figure A.1.13: A JDOQL query, requesting all Entities with attached sensors that send their data in `KLM_Point` format.

Context Retrieval

When context had been stored, it is possible to query it by the usage of the `IContextConsumption` interface. Either single-line JDOQL queries or the n-dimensional context retrieval could be used. The example code in A.1.13
requests all Entities that contain a ‘6’ in their id and have an attached InformationSource instance with Representation KML_Point. Figure A.1.14 shows how the context retrieval method getEntitiesByContext(filters, entitySubclass, referencePoint) is used. In this scenario, a four dimensional context retrieval is conducted.

```java
/**
 * create a context based close neighbors search for 3 aspects (4 dimensions)
 */

// location
ContextFilter firstFilter = new ContextFilter("location");
ContextCriteria criteria1 = new RangeCriteria("KML_Point", "longitude",
    new Double(-100), new Double(100));
ContextCriteria criteria2 = new RangeCriteria("KML_Point", "latitude",
    new Double(-180), new Double(180));
firstFilter.addCriteria(criteria1);
firstFilter.addCriteria(criteria2);

// velocity
ContextFilter secondFilter = new ContextFilter("velocity");
criteria1 = new RangeCriteria("KMH_Proprietary", "km/h",
    new Double(0.2), new Double(0.4));
secondFilter.addCriteria(criteria1);

// orientation
ContextFilter thirdFilter = new ContextFilter("orientation");
criteria1 = new RangeCriteria("Degree_Proprietary", "degree",
    new Double(20), new Double(270));
thirdFilter.addCriteria(criteria1);"`.Figure A.1.14: An example of how the optimized entity retrieval engine works, when applied to a concrete model.

### A.2 Ad-Hoc History Retrieval

A kindergarten class is making a trip into the mountains. For security reasons, each child is equipped with a GPS receiver and an up-link to the context-base (e.g. a mobile phone) that frequently reports in its position; just in case someone gets lost. After a while one of the teachers, in shock, finds out that a child is missing. She tries to reach it by phone, but it is turned off. So the teacher rings the police, knowing that they have access to the global context-base. The officer is able to track the kids locational-scope down. This was possible due to the GPS data that had been stored until the kids phone went offline, a while ago. Since the context-base as well contains the teachers context data, it is a simple task to guide her towards the environment where the child disappeared.

Figure A.2.15 presents the use case, considering a global context-base with the police having access to all the context data. The police officer is the consumer of the ad-hoc service, which uses the context-base for retrieval.
A.2.1 Ad-hoc Service

Such a service would require that either a graphical user interface for retrieval support is present at each police station, or that each police station has an expert for JDOQL retrieval. In the latter case there is no need for further development, looking at the context-base how it is today. A means to plot paths from history data on maps would be necessary for a graphical understanding of where individuals are.

A.2.2 Domain Model

This scenario has no new domain model requirements, assuming that the context-base contains the traffic congestion model extension. It might anyway be useful to model Person as a concrete subclass of Entity, so to get semantic distinction between the concepts.
The following itemization shows the required concepts:

- **Entity**: Person
- **InformationSource**: Sensor, GpsSensor
- **Aspect**: Location
- **IAспектComparator**: LocationComparator
- **Representation**: KMLPoint (Google Maps)
- **RepresentationMapper**: KmlMapper

So if the context-base contains all model extensions for the previously presented traffic scenario, only the Person class with its JDO descriptor has to be uploaded. Now the teachers, as well as the kids and their sensors can be virtually created.

### A.3 Messenger Brokering

For this scenario it is assumed that two messengers and an online community are integrated. It can be further assumed that the messengers are Skype and ICQ, where Facebook serves as the community. Whenever a user changes the presence state in Skype it should be reported to his buddies and vice versa. A buddy does not need to use Skype explicitly, but could use ICQ or Facebook. A brokering between these protocols would bring along that features from n different protocols for overlapping functionality can be used by deciding for only one of them. Since there are many messengers and protocols, this would mean that a larger community can be wired directly.

Figure A.3.16 shows the use-case. The three users on the left use the service of their brokering application, which runs in the background. The use case shows that the service might involve a cross-domain presence state distribution, but as well the translation of messages themselves. That would bring along the complete chat functionality across platforms.

#### A.3.1 Brokering Service

Since the general functionality is very much alike for these applications, the main differences lie in the syntax of messengers and the protocol. The application developer decides at design time which one of the three platforms structures will be used as the standard **Representation**. He will probably decide for the most expressive one.

The hard work now is to make the inter-lingua translation pattern work by the implementations of **IRepresentationMapper** for each platform. When that is achieved, the application integration could be done on the application layer by fetching, translating and forwarding port packets. The same applies for the presence state updates. The process description in A.3.17 just focuses on the presence state distribution. It is triggered by a local event and its proprietary message is intercepted, interpreted, translated and forwarded to all subscribers for all three platforms.
Figure A.3.16: The messenger brokering use case. The brokering service in the center works in the background to translate and delegate context updates.

Figure A.3.17: The brokers event-driven working-process for presence status updates.
A.3.2 Domain Model

The following enumeration contains all the concepts, having to be included in the context model extension:

- **Entity**: Person
- **InformationSource**: PresenceSource, IcqPresenceSource, SkypePresenceSource, FacebookPresenceSource
- **Aspect**: Presence
- **IAspectComparator**: PresenceComparator
- **Representation**: Icq, Skype, Facebook
- **IRepresentationMapper**: IcqMapper, SkypeMapper, FacebookMapper

Where in former examples each Aspect only has one particular Representation, this whole scenario builds upon the idea that there could be many of them. However, there is not more to develop for each Representation than in the other scenarios.

**Summary** In this chapter three solutions for problems with relevance to real life applications were presented, proposing the CII-model and implementation. It shows that the usage and especially the reuse of concepts in the context-base makes the development comfortable, while remaining a very broad and flexible scope. The biggest advantage is that all these domain models can be managed in one context-base. That enables for service developers to create services combining context from different domains, what allows arbitrary context integration considering traffic, messenger and history data, for innovative services.
B Tutorial

For the context-base to be made available to a system there is some installation work to be done. Appendix B will shortly introduce, how to

1. setup the environment,
2. create the database &
3. administrate it.

The chapters in this document are supposed to be read in the order they are presented. If you are just interested into making the database work, you can leave out the reading of section four, which gives deeper insight into the actual connection between all the setup files and folders.

Pre-requirements

For the database to work, the following has to be installed on your system:

1. Sun Microsystems JDK, in version 5.0, ’Tiger’ (or higher),
2. Apache Ant in version 1.2 (or higher),
3. A database of the following type:
   - RDBMS: including the JDBC driver (recommended approach)
   - ODBMS: currently db4o

For the database connection, a so called access platform is applied; in our case DataNucleus. It abstracts from the concrete database structure, providing one general Java interface and a standard retrieval language, JDOQL.

B.1 Step by Step Guide

In this section I will address the steps for setting up the context-base. Everything required to successfully install the database can be found on the MediaSense SVN server. The mentioned SVN paths start out from ContextBase/code/se.mium.mediasense.shared.context.base/. This tutorial considers, that you are using a RDBMS, possessing the a suiting JDBC driver.

B.1.1 Creating the Database

1. ensure that you system fulfills all the pre-requirements.
2. download the folder
ContextBase/code/se.miun.mediasense.shared.context.base/
from SVN to your machine. This folder will from now on be referred
to as your base/ folder.

3. copy your JDBC driver to base/lib/.

4. open the file datanucleus.properties in your base/ folder.

5. modify the following properties to fit your JDBC installation:
   - javax.jdo.option.ConnectionDriverName
   - javax.jdo.option.ConnectionURL
   - javax.jdo.option.ConnectionUserName
   - javax.jdo.option.ConnectionPassword

6. open the build.xml file in your base folder and double-check that the
   versions of the property by name model.jar matches the correspond-
   ing jar file in your lib/ folder. If not, modify the build file entry to
   match the jar file version.

7. open a terminal and traverse to your base folder.

8. run the ant script create.environment by the executing
   ‘ant.create.environment’. A directory-tree with root base/context/
   will become created.

9. execute 'ant createschema' to create the basic concepts (tables) in your
   database.

If both scripts finish up by a final print of BUILD SUCCESSFUL, you should
be able to verify that the basic concept tables were created by applying your
favorite database administration tool of choice.

B.1.2 Creating the Application

Now when the context-base is installed, you might be interested into devel-
oping an application, that makes use of it’s features. The only recommended
ways to access it are implementations of the interfaces
se.miun.mediasense.context.api.IContextConsumption
and se.miun.mediasense.context.api.IContextSpecification specified
in se.miun.mediasense.context.api-*.jar. The proposed implementa-
tion resides in se.miun.mediasense.context.persistence-*.jar.

Two things are of importance here:

1. How do I setup my classpath to suffice the context-base service?

2. How do I receive a service instance from within my Java code?
Setting up the classpath

1. ensure that you have successfully finished the tutorial *Creating the database* (B.1.1).

2. add all jar files under base/lib/ to the build-path of your application.

3. add the folders base/context/target/domain and base/ to the runtime-classpath of your application.

4. create a file contextBase.properties under the root of your application.

5. put 2 entries into the file like the following:
   - contextBase_HOME =CONTEXT_BASE_HOME (path of your base/ folder), and
   - historyMode=MODE (true or false; depending on if you want history data to be stored)

   e.g:
   ```
   contextBase_HOME =home/peter/contextbase
   historyMode=true
   ```

The classpath should now be valid, so that you can use the service from within your application.

Accessing the context-base services

A service instance is received by a static call of the methods in se.miun.mediasense.context.persistence.ContextBaseFactory by name:

- `getContextConsumption();`
- `getContextSpecification();`

Therefore the actual call will be for instance `ContextBaseFactory.getContextConsumption();` from within your Java code.

B.2 What is What

In this section I will shortly explain all the files necessary for the setup and their value for the application. The itemized entries can be found in ContextBase/code/se.miun.mediasense.shared.context.base/:

- lib/
- build.xml
- datanucleus.properties
- log4j.properties

Their use will shortly be explained.
B.2.1 lib-folder (1)

In the lib folder, 3 self developed jar files reside. Table B.1 enumerates them. These jar files have to be in the classpath, so the application can function. Image B.2.1 visualizes their dependencies.

Where the model and api jar don’t have any dependencies to the libraries in the lib/ folder, does the persistence jar entirely build up on the access platform DataNucleus and it’s libraries. All other lib/ jars have to be present in the classpath, so the persistence framework can do it’s job (delegating queries and converting formats).

B.2.2 lib-folder (2)

There are nine other jar files in the lib/ folder, of which all are of importance for the application development. Table B.2 summarizes their contributions.

B.2.3 files

The base/ folder contains the important setup files for the context-base. A list with explanations can be found in table B.3. build.xml is the script, which is called from within the code, for the runtime integration of classes, or creation of tables. It contains some useful tasks for general deletion and creation of the database as well.

The datanucleus.properties carries the main setup for database related setup. There are many screws here to fasten in terms of performance. A full description of this can be found in

http://www.datanucleus.org/products/accessplatform_1.0/persistence_properties.html

Table B.1: The context-base jar files under lib/.

<table>
<thead>
<tr>
<th>file</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>se.mium.mediasense.context.model-*-jar</td>
<td>the model classes</td>
</tr>
<tr>
<td>se.mium.mediasense.context.api-*-jar</td>
<td>the application programming interface</td>
</tr>
<tr>
<td>se.mium.mediasense.context.persistence-*-jar</td>
<td>an implementation of the API</td>
</tr>
</tbody>
</table>
Table B.2: The third party jar files under lib/.

<table>
<thead>
<tr>
<th>folder</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>ant-launcher.jar</td>
<td>a processor that enables ant task execution from within java</td>
</tr>
<tr>
<td>ant.jar</td>
<td>the core ant classes</td>
</tr>
<tr>
<td>asm-*.jar</td>
<td>small byte-code manipulation framework</td>
</tr>
<tr>
<td>datanucleus-core-*.jar</td>
<td>core of the access platform DataNucleus</td>
</tr>
<tr>
<td>datanucleus-enhancer-*.jar</td>
<td>the DataNucleus tool to enhance classes for persistency</td>
</tr>
<tr>
<td>datanucleus-rdbms-*.jar</td>
<td>the framework that deals with RDBMS as backend</td>
</tr>
<tr>
<td>jdo2-api-*.jar</td>
<td>the Java Data Objects standard interfaces</td>
</tr>
<tr>
<td>log4j-<em>.jar-</em>.*jar</td>
<td>the logging tool, used by DataNucleus</td>
</tr>
<tr>
<td>tools.jar-*.jar</td>
<td>the sun tools library, providing plenty of services</td>
</tr>
</tbody>
</table>

Table B.3: The context-base setup files under setup/.

<table>
<thead>
<tr>
<th>file</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>build.xml</td>
<td>the script file, automating many context tasks</td>
</tr>
<tr>
<td>datanucleus.properties</td>
<td>the java properties file for the setup of the access platform</td>
</tr>
<tr>
<td>log4j.properties</td>
<td>the java properties file for the tweaking of log4j logging</td>
</tr>
</tbody>
</table>

It also contains the setup for the database meta-data, like explained in B.1.1.

An interesting property is for instance the log4j.logger.DataNucleus flag in log4j.properties. For a test environment this might be set to DEBUG, due to richer bug reports. But in user environments, this should be unset, for a better performance (change DEBUG to INFO).

**Summary** A description for the setup of the context-base had been given, which should enable a Java application developer to start developing context-aware applications.
Making a software extensible and enhanceable is one of the most important aspects in software design. It is in general not sufficient to only follow a clear software design, but to as well document it. This chapter has the purpose of presenting the context-base design to developers and to everyone considering it crucial for further investigation of it’s applicability.

After addressing the systems components, the data flow and a short overview about how the shared code-base is currently managed is presented.

C.1 Components

Three components make the system: model, api and persistence. Their dependencies are shown in B.2.1. model.jar is a Java implementation of the CII-model. It contains all the concepts as Java classes. These classes are enhanced by the DataNucleus enhancer, which makes regular Java objects storable from within the DataNucleus access platform. api.jar only contains the specification of the two developer interfaces and a small framework for the definition of filters, when applying the n-dimensional context retrieval. The persistence component is the implementation of api, depending on both, model and api.

All components are entirely written in Java, using Apache-Ant for build processes at deployment and runtime.

C.1.1 Model

The model is the ontology into which all domain-concepts are integrated. Like shown in appendix A, Use Cases, new storable sub-concepts are created by extending the classes. The implementations of the interfaces IAspectComparator and IRepresentationMapper are not of a storable type, since they are to be entirely stateless and therefore only remain in the global classpath. They are activated by a technique called reflection, which allows the platform to instantiate objects of types unknown at build time, but specified by class identifiers at runtime.

In figure C.1.1, the model’s UML class diagram is shown. Similarities with the model in 2.3.4 are obvious. What can be seen is that the IAspectComparator is a mandatory part of the model, serving to keep entries in order inside of the context-base. To make the context principles easier to explore, an additional Description class had been added. Each
Representation holds a Description, so that every consumer is able to understand and reuse Representations, without having to understand a whole domain model.

### C.1.2 API

The two API interfaces have been presented in chapter three. The framework for n-dimensional context retrieval requires a more thorough explanation. The example code in appendix A should make clear how to use it. Anyway, a more precise description follows here.

Figure C.1.2 shows the filter package. Every ContextFilter filters Entities, representing one Aspect. Each of them is aggregated by a collection of ContextCriteria. A criteria defines a range for one Dimension. This range could either be just a value (ValueCriteria) or a whole range (RangeCriteria). All criteria are concatenated for this filter by a logical AND. At the moment it is not possible to filter for two values or ranges of one dimension in a single query. This could be improved in later versions.

An array of these filters is becoming applied for each context search. They are as well concatenated by logical AND’s. All of them have to be fulfilled for a ContextValue to qualify as a valid result.

### C.1.3 Persistence

All controller work for the creation and maintenance of the context-base is stored by the persistence module. The conceptual class-design can be seen in the UML diagram C.1.3.

The api interfaces in the diagrams center are implemented by Context-Consumption and ContextSpecification respectively. These two classes delegate the calls with a freshly created database-connection to the respective methods of the super classes ConsumptionEnabler and Specification-Enabler. This is done to optimize connection reuse and therefore performance for tasks that include more than one database access. These two share the basic platform access code and both extend the class DatabaseEnabler. EntityContextEnabler is the class in the Consumption hierarchy, dealing with the n-dimensional context retrieval.

The HistoryContextConsumption is an extension of ContextConsumption. It manages the recording of passed context values. If the developer specifies to store history data, the HistoryContextConsumption takes the position for the class ContextConsumption at runtime. This option can be set in the file contextBase.properties. Implementations of the context interfaces are accessed by static methods in ContextBaseFactory, following the factory design-pattern. This is where the developer should seek his context-service instance, not from anywhere else.

### The Real Data

The developer only deals with Java objects. How context is really stored is for the time being a relational database. In the future this could be changed to an object oriented database, but their current drawbacks in scalability
Figure C.1.1: The class model of the presented CII-model implementation in java.
Figure C.1.2: The context filter framework for n-dimensional context retrieval.

Figure C.1.3: The context controller class hierarchy.
incapacitate them as a choice for this thesis. The PostgreSQL database schema can be seen in C.1.4. The management of data-type mapping is the access platforms task.

The persistence layer creates code and meta-descriptors on the fly. These are integrated into the context-base by applying DataNucleus schema-tools. DataNucleus schema-tools is an Apache-Ant based tool for the management of relational databases. It can be used from within Ant build files or from within Java code. Informal, the extensions are of four different types:

1. new sub-concepts of InformationSource: new InformationSource sub-table (developer defined) is created. Each InformationSource of that sub-concept is stored in this table.

2. concrete InformationSources: new History ContextValue table (dynamically defined) is created. Each time a ContextValue becomes updated, the old one is getting archived into the history table of that specific InformationSource. Each InformationSource has a history table if the history option is marked.

3. Aspect: new ContextValue table (dynamically defined) is created. All current ContextValues for that Aspect are stored in there.

4. new sub-concept of Entity: new Entity sub-table (developer defined) is created. All entries of that sub-concept are stored in this table.

The persistence framework ensures that all ContextValues end up in the right tables. This is done to a large extend by the application of Java's reflection mechanism.

**Executing the Job**

Depending on the task to be conducted, different background activities are triggered for the integration of context. In this section the most important
tasks are depicted. Their flow is visualized by UML flow-diagrams.

**Uploading a Class** Uploading a class is a rather simple process (see C.1.5). The code creation is on the developers side. He uploads either only the Java file (non storable concepts) or both, Java file and the JDO descriptor (storable concepts). The class, serialized as a byte array, is transferred to the ContextSpecification instance. ContextSpecification stores the Java file in a temp folder, compiles it, and copies it into the domain-model folder. Afterwards the intermediate file in the temp folder is becoming deleted. Since the domain-model folder is part of all applications classpath, this enables the usage for all context-aware applications.

An example of such a process is the upload of an implementation for the interface IRepresentationMapper.

**Using JDOQL queries** This task is as well quite simple (C.1.6). The persistence layer work is essentially to forward the query from the user to the access platform and the way back. Exemplary this task shows how the access platform is getting involved. The ContextConsumption instance at creation time retrieves an instance of type PersistenceManagerFactory from the access platform. This factory’s obligation at runtime is to make so called PersistenceManager available for the consumers. Hence it creates a PersistenceManager and returns it to the caller. The PersistenceManager can be compared to a JDBC-database connection. The caller is responsible to start transactions, end them and close them to release system resources. That is what happens in this flow diagram: a transaction is getting started, the query forwarded, executed and the result send back to the user of the framework, after the PersistenceManager had been closed.
Figure C.1.6: The flow diagram of a JDOQL query, executed on the API user side.

Figure C.1.7: The n-dimensional context retrieval.
Retrieving Entities Since the n-dimensional context retrieval involves more cascaded tasks than the former presented methods, some of the activities in the diagram are just explained and not fully revealed. It can be seen on picture C.1.7 that a PersistenceManager is pulled from the PersistenceManagerFactory and a transaction is started. For an optimized performance, the optimal filter is chosen.

The database call firstly retrieves the potential values, applying the best filter. Afterwards all Entities that hold the respective InformationSources are retrieved. By applying the filters, they are reduced to the ones relevant for the query. When the candidates have been found, they are all ordered according to the reference point, as explained in Chapter 3, Architecture and Implementation.

Adding InformationSources The task involving most activity is the addition of an InformationSource at runtime (see figure C.1.8). The InformationSources are stored by makePersistent(is). It follows a sub-tasks, which creates a history table by reusing the persistenceManager. This history table is dynamically created from the persistence framework, which ensures that all outdated context values are timestamped and stored in there. This sub-task invokes the creation of a Java class and a meta-descriptor. After the creation, they are uploaded to the context-base by uploadClassToContextBase.

The class then gets compiled. Now, the DataNucleus enhancer tool enhances the created byte-code under the support of the JDO meta-descriptor. The schema tool creates the related database tables and finally the concept is dynamically added to the classpath for global availability. Picture C.1.9 shows how dynamically created context values look like for the example in the use case appendix. These sub-concepts of ContextValue are created for each Aspect, which is added to the context-base.

C.2 Domain Context Management

Like shown in the context controller figure 3.1.2, the context-base contains a code-base for the storage of the byte-code and a context-database, in which all context remains. Since the database is managed by the access platform and the code-base is not, it is important to explain how the code-base works. Figure C.1.10 shows the code-base, starting out from the context folder which resides in the base folder of the shared context-base of the network. It follows a description of the folders meanings:

- **domain** contains all domain-level concepts that ever had been added to the context-base. They reside here in the uploaded uncompiled Java and JDO version and just serve for the purpose of documentation. It is possible for each developer to verify if a concept exists. The existence of the domain folder has no impact on the application itself; there are no dependencies.

- **meta** is the place where the JDO meta-descriptors for the general CII components are stored. The preferred way to store them would have been in the model.jar file, but due to some complications with the runtime access from within Apache-Ant, the chosen approach seemed to be appropriate. The meta folder should never be touched.
Figure C.1.8: The whole chain of events that is triggered, when an InformationSource is added to the context-base.
Figure C.1.9: The context classes for the traffic information scenario, created at runtime from the context-controller.

Figure C.1.10: The code-base folder structure.

**target/classes** is the standard folder, which DataNucleus uses for the storage of class files. It is not explicitly used by the persistence framework, but might be needed for DataNucleus to work properly.

**target/domain** is the most important of the context folders. In this folder the compiled and enhanced byte-code versions of all domain-model extensions reside. It contains therefore the executable versions of all *.java classes in the domain folder. This folder has to be part of the classpath for each context-aware application, so that dynamical changes to the classpath are directly available for all other context-aware applications.

**temp** is the folder which is used as an intermediate storage for all files that are integrated into the context-base. After each job, this folder is emptied.

**Summary** The purpose of this chapter was to give a short software specification of the context-base. It is still much work left for this framework to be ready for real world application, but the first steps towards a functional implementation are made.