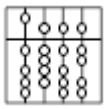
Technische Universität München
Fakultät für Informatik



Master's Thesis

Spatial Context Management for Augmented Reality Applications

SCORE: Spatial Context Ontology Reasoning Environment

Jan-Gregor Fischer, B.Sc. (TUM)

Aufgabenstellerin:	**Prof. Gudrun Klinker, Ph.D.**
Betreuer:	**Dipl.-Inf. Marcus Tönnis**
Abgabedatum:	**15. Mai 2005**

Ich versichere, dass ich diese Master's Thesis selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 13. Mai 2005 Jan-Gregor Fischer

Danksagungen

Frau Professor Gudrun Klinker danke ich für die Überlassung des Themas sowie für ihr jederzeit freundliches Entgegenkommen.

Für die Anleitung und Unterstützung bei der Durchführung dieser Arbeit gilt Herrn Tönnis mein besonderer Dank.

Schließlich möchte ich mich herzlichst bei meinen Eltern und meiner Schwester bedanken, die mir während dieser Arbeit stets Rückhalt und in besonders anspruchsvollen Zeiten neuen Mut gaben.

Abstract

The thesis describes the development of the Spatial Context Ontology Reasoning Environment (SCORE), that supports the management of explicit and implicit spatial context for Augmented Reality applications.

Augmented Reality (AR) forms the link between reality and Virtual Reality by augmenting the user's environment with virtual objects and information, that can be interacted with. Here in general head-mounted displays are used to present the augmented view to the user.

This research field is investigated at the chair for Computer Aided Medical Procedures and Augmented Reality at the Technische Universität München. Among many other things it covers new user interaction paradigms, sensor analysis and the access to structured information in intelligent environments. A common claim to Augmented Reality systems is not to restrict the users' scope of motion. For supporting mobility the DWARF framework is frequently employed. It represents a distributed AR framework providing ad hoc interoperability of services, that self-assemble into AR systems.

AR applications must efficiently handle spatial information of real and virtual objects in order to provide a “natural” feeling of the augmented world to the user. In addition to the mere location-awareness they involve situational aspects that are related to the user more and more. As a matter of fact the management of spatial context increasingly gains importance to applications in AR.

This thesis presents a service-oriented framework that is aimed at facilitating the integration of these aspects. It follows a novel approach in spatial context management, that incorporates traditional coordinate-based context models and contextual ontologies that are well-known from research efforts with respect to the Semantic Web. Thus efficiency is combined with a declarative knowledge representation enabling knowledge sharing and reuse.

Besides the explanation and discussion of the general research topics, the work is based on, the thesis presents a first application using the presented framework. It is a proactive safety system for vehicles, and is part of an interdisciplinary research project for user-centered driver assistance, that is supported by techniques of Augmented Reality.

Table of Contents

List of Definitions

List of Figures

List of Tables

List of Examples

1. Introduction

This first chapter introduces the fundamental background of my thesis. It also addresses readers who are not already familiar with the term “Augmented Reality”.

In the scope of my thesis I designed and implemented software that aims to facilitate the management of spatial context for Augmented Reality applications. This software uses the DWARF framework – a research project for mobile Augmented Reality – for validating the investigated concepts.

Besides these topics I will explain the particular problem, that is addressed by this work, in the following sections.

1.1. Motivation

With the increasing spread of wireless mobile devices and new innovative applications in the domain of location-aware computing the processing of spatial knowledge became more and more important in recent years. Today mobile phones enable users to access the Internet, play games with people they never met, shoot pictures and films without carrying a dedicated camera, or use navigation and routing information to visit unknown locations without an antiquated city map. And this is possible in addition to the original task of mobile telephones: that is telephoning. As for mobile phones the ever increasing computation power of mobile and wearable devices such as car navigation systems, lightweight sub-notebooks and PDAs also enable applications which require to access spatial information efficiently. For example, this spatial information may be needed to find out where the user is located, in which direction she looks, which objects are near to her and therefore can be accessed or interacted with.

The amount of physical quantities like position and orientation data, velocity and acceleration, or distances to other users and objects sensed in the real world increases continuously within the domains of more and more complex applications, that make use of such spatial information in their computations. From the view of the application developer their is also often a need to keep track of structured spatial data and compare it to related information acquired within a former period of time. For example, an interactive tour guide application could offer the user, who comes to Berlin for the first time, to visit the modern art exhibition as it knows that she is interested in modern art and has seen a similar exhibition in Munich two weeks before.

Also applications often have to decide rapidly on the basis of spatial information to help or protect the user. Imagine a driving assistance system which implements an automatic emergency break. The system should assist the driver in dangerous situations in which she is definitely not able to react in time to avoid a crash with her car. The spatial information, the system's core algorithm is based on, which has to decide whether such a situation takes place at the moment or not, must be reliable and most up-to-date. Computations on faulty or obsolete spatial information may lead to a crash where none would happen if the driver went without this assistance system.

Among others, applications which are based on the user's location use spatial information

which regard the user and other objects in the real world to reason about it in order to deduce enriched information. For example, imagine you are driving with your car on the right lane of the motorway, and you want to overtake a slow lorry in front of you at present. You set the left trafficator and want to change lanes. Since you simply forgot to check the rear-view mirror, you will overlook the fast sports car appearing on the left lane behind your vehicle, and you are just risking to collide with it. Let's assume your automobile is equipped with an assistance system that should help avoiding such situations. Directly after it gets the information that you activated the left indicator, it analyzes the spatial vehicle data provided by sensors in your car, and retrieves the spatial information for the sports car for instance from a rear-mounted radar sensor. Then it would deduce that the sports car is not only an obstacle for you, but that it is a fast driving automobile reaching your relative position on the left lane in exactly 0.36 seconds, so that the sport car's driver will not be able to break or avoid your car in time for averting the crash if you continued to switch to the left lane. So directly after you set the trafficator your driving assistance system would inform you by an optical, haptical or acoustical alarm about the obstacle appearing from the rear left, and you would immediately break off the overtaking procedure.

Besides this dramatic but certainly useful alarming it is also well known that in almost all application domains the intended user expects to be informed about an application's current state and task. For example, the tour guide mentioned before should tell the tourist about places in the city which are worth seeing. The automatic emergency break assistance system should alarm the driver that it just took control with respect to the breaking systems of the car. And the system in the last example should output an alarm if it is not safe to overtake other vehicles.

It is a common procedure that user information are displayed on a screen the user has in front of her. While the number of applications increases more and more information are placed on this display. Besides the fact that a driver may overlook an important crash alert, because she concentrates on the traffic and not on the display, the crush of information does not contribute to an intuitive user output interface. To stay with this particular example it would be a lot more convenient if the important (crash avoidance) information would be placed directly in the direction the hazard is sourced. Such information may be a big red fast blinking virtual arrow which is located directly in the field of the user's view, and points to the sports car which is the obstacle in the example mentioned before. If the user moves the head the blinking arrow would stay at the same position relative to the user's direction of view, so that she is able to recognize it independently from where she is currently looking at.

By adding the synthetic arrow to the real world a so called *Augmented Reality* is created where the user experiences additional virtual information which are associated with real things.

1.2. About Augmented Reality

Augmented Reality (AR) forms the link between reality and Virtual Reality (VR). Virtual Reality applications are meant to simulate reality, so for example it is possible to train pilots in flight simulators. But when piloting a real airplane one can only fall back on the lessons learned in the simulator. Instead of moving the user into a completely virtual world, Augmented Reality systems enrich the user's view, and therefore her real surroundings by adding virtual information to it in real-time. That means, the combination of virtuality and reality is efficient enough so that the user gets a “natural” impression of the augmented world.

The purpose of AR is to support people with virtual data when they are interacting with real things. So the pilot may be supported by a virtual airplane which is located in front of her and represents the position and orientation of the own real plane as it will be localized in three seconds. Therefore the pilot could use this synthetic information to control a correct approach towards the runway.

Characteristics of AR *Augmented Reality* forms a ***combination of reality and virtuality***, where this composition must be able in ***real-time.*** In addition to these requirements Azuma, who defined characteristics of AR [AZU95] in 1995, states that the mere augmentation of the real world is not enough to form Augmented Reality. Instead the virtual items have to be ***registered in 3D***, which means that the virtual model of the real world is 3-dimensional, and that virtuality is continuously modified whenever reality changes. In addition to this, ***interaction*** with the virtual objects must be possible in real-time. This requirement evolves a need for novel user interface devices which must be easily usable in the three-dimensional space [OVI02]. For example, films like “Men in Black” or “The Matrix” which include virtual elements are not interactive at all, because the user can not modify any parts of these films, no matter how often one is watching them, and therefore do not satisfy this characteristic of Augmented Reality.

Why should Reality be augmented at all? To answer this central question the importance of information and information exchange has to be analyzed. People are curious all the time and want to know more about their surroundings. Either at work or at leisure time they try to gain more and more information often without noticing this urge for knowledge. Already in childhood children ask a lot of questions starting with the words “Who”, “Where”, “When”, “What” and “Why” like for example “What is this over there?”. In school students learn from teachers and school books above all. Later when at work, people need to know the work flows and the characteristics of the products they create or services they provide. Often this knowledge has to be adapted when new and unknown situations occur.

For example, a mechanic, who has to maintain vehicles whose various electronic parts are developed into more and more complex systems, has to improve hers knowledge continuously to keep track with the new technology. As another example physicians have to take the patient's individual mental and physical constitution into consideration before and during the treatment. They also have to continue their's education all the time by reading scientific arti-

cles and books or by attending events for further training.

The information people are curious about may be bound to other people, their knowledge, history and their experience. It can also address animals or certain objects like vehicles or parts of them. It may cover important data which people can not detect with their own senses, or which is provided much faster by a computer than by experiencing on one's own. A car crash avoidance alert is an example for the latter one. The way of representing the information to other people is also varying a lot. Information overflow and the problem with extracting the information necessary at the current moment are only two reasons which enforce researchers to develop software systems whose task is to help people to deal with reality.

Optical and Video see-through Head-Mounted Displays The most common augmentation method is visual augmentation. Although it is possible to display the augmented scenery on portable computers like handhelds or by projection with video beamers, the user experiences a more intuitive and realistic impression of the unadulterated surroundings if she is wearing special glasses, the so-called head-mounted display (HMD). An example for this setup is shown in figure 1, where a car designer is using Virtual Reality Aided Design (VRAD) to get a quick impression of the product already in an early stage of it's development.

Figure 1: A virtual car in front of a user's desk

Fata Morgana project at the Technische Universität München

The user – a car designer – can develop virtual product mock-ups and view them through an optical see-through HMD by rotating and moving the mock-up before a real model of the car is manufactured at all.

Two types of head-mounted displays can be distinguished. Using the ***optical see-through HMD*** the user is able to see the real world trough a semi-transparent mirror by looking straight through it. A display shows additional virtual information by mirroring it to the user's eyes. To know where to insert the synthetic objects, the position and orientation of the user's head must be calculated with the help of a tracking system [RBG01]. The optical see-through HMD provides a relatively natural feeling to the user as she is still able to see the real environment, also if the brightness is dimmed due to the transparent mirror. However these glasses create a “swimming” impression of the virtual objects since these are rendered with a small but noticeable delay to the user's sensing of the real world. On the other hand high resolutions and the easy using due to small proportions are their main advantages.

The other type of glasses are covered by the ***video-see-through HMD***s. They also include a mirror like the HMDs before, but the difference is that this mirror is not transparent but opaque, so that the user can not see the real surroundings directly. Instead, a camera, which

is attached to the user's head, films the area the user is just looking at, and analyzes the captured image to estimate the user's pose. Then the virtual information is added to the filmed reality at calculated positions, and the modified pictures are then displayed to the user. Generally these glasses support lower resolutions than the optical see-through HMDs do. Also for each picture, displaying the real part of the world has to be paused until the virtual part is rendered. Therefore the person wearing these glasses has to get used to it.

For a more detailed comparison of optical and video-see-through HMDs I would like to refer to [RF00].

Applications for AR Since Azuma defined the characteristics of Augmented Reality a lot of new applications have been developed in this field. The various application domains include among many other things:

- *manufacturing and repair* – Mechanics still have to study manuals in paper form and compare photographs and drawings to real parts which are hidden somewhere in cars, trucks, airplanes or ships. It would be more comfortable to be guided through the complete repair process with the help of interactive maintenance instructions. The repair steps are depicted on the mechanic's HMD in detail by augmenting each real part, that is subject to the current repair task, with virtual information.

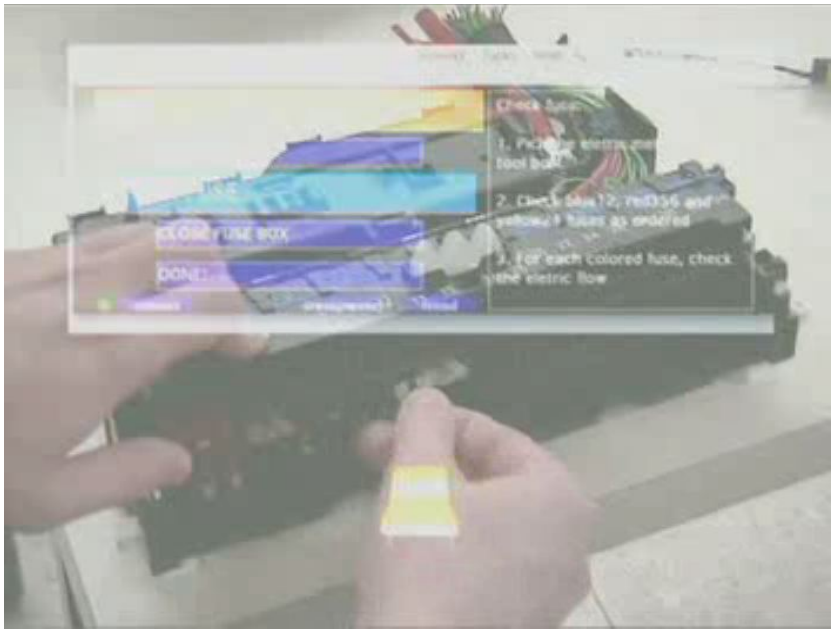
  

  Figure 2: A mechanic's view onto a broken part through the HMD

  The user is guided trough the repair process with the help of virtual repair instructions.

  During the winter term 2001 at the Technische Universität München (TU München) the “Traveling Repair and Maintenance Platform” (TRAMP) was developed addressing this AR domain. Figure 2 shows the augmented repair process from the view of the mechanic.

- *games* – ARQuake[1] for instance is an AR research project developed at the University of South Australia since 1998. This game combines the virtual reality of the popular “first-person shooter” Quake with players and outdoor / indoor environments in the real

1 The official ARQuake Web site can be found at “http://wearables.unisa.edu.au/arquake”.

world [PT02].

- *sightseeing* with virtual historical information and augmented view of buildings that are ruins in reality
- *emergency situations* – displaying emergency paths to endangered persons who try to escape from buildings
- *military* – information about friendly and hostile troops to decrease the threat of friendly fire
- *surgery* – virtual display of the patient's interior organs which may be occluded by tissue above
- *education*– Augmented Reality systems can be used to help people training spatial abilities like mathematic and geometric knowledge.

To learn more about the different fields of AR applications the interested reader is suggested to visit the Augmented Reality projects overview[1] at the Technische Universität München in the Internet. Another interesting web site is the one of Studierstube[2], a framework which helps developing Augmented Reality applications. Studierstube has been developed at the institute for Software Technology and Interactive Systems at the Vienna University of Technology since 1996.

An important claim to Augmented Reality systems is the integration of distributed systems, which support mobile devices and dedicated services to be used on demand. AR needs a lot of computation power that can still not be procured by small mobile devices. On the other hand users certainly do not want to carry huge computers with them, which are powerful in computation but also very heavy and obstructive. The solution for this problem are distributed systems connected via wireless networks. So services requiring a lot of computing performance or which are not available directly to the user can be run on nearby high performance computing systems. This setup is crucial to Augmented Reality systems and has an important need to an underlaying distributed communication infrastructure.

The following chapter discusses such a software infrastructure, which has been designed and implemented at the TU München since 2000.

1.3. Why is that DWARF always following me?

Among other things the title of this chapter refers to DWARF, the Distributed Wearable Augmented Reality framework, which is employed in many Augmented Reality projects at the TU München, and is also used in the scope of this thesis. It represents the basic framework for distributed communication and *ad hoc* interoperability of services where each of them provide particular abilities that can frequently be reused in various AR applications.

The system design of the DWARF infrastructure is described in detail in [MAC01].

1 http://wwwbruegge.in.tum.de/DWARF/ProjectsOverview
2 http://www.studierstube.org/index.html

MacWilliams states on pages four and five:

"One of the goals in designing DWARF was to consistently use ad hoc services, so that the system is essentially self-assembling. This makes it possible to distribute different computing tasks onto different hardware devices which the user can then simply plug together to make a working system. It also means that separate mobile systems and intelligent environments using the same framework will be able to cooperate easily – indeed, they will spontaneously self-assemble into a larger system."

Since its first demonstration in December 2000 (as a campus navigation system) DWARF has been further developed and improved.

The framework essentially consists of so-called *DWARF services* and the distributed *DWARF middleware*. A number of services which are dynamically composed and connected by the middleware form an architecture of *subsystems* allowing various Augmented Reality applications to be built. These concepts are explained in the following sub-chapters. A more detailed description and definition of frameworks as a pattern of designing software architectures is provided when explaining the tasks and focus of context management frameworks later in this thesis.

1.3.1. Services

A service in the context of DWARF is a software process running on some computing system. For example, it can be a tracking service whose only task is to tell about the user's position in a room. Each service is registered with the DWARF middleware (described in the next section) and is able to communicate with other services running on the same computer or a distant one generally using TCP/IP[1] compatible networks like cable-bound Ethernet or for example wireless LAN[2]. The service may be dependent on information which is provided by others and may place information at other service's disposal. This concept of *needs* and *abilities* of information is configured in a service description and specifies the kind and amount of other DWARF services which can connect to the one the service description belongs to. The needs and abilities are defined by the use of types telling the framework which services are able to communicate on the same information type's basis. These types represent the interfaces for service communication.

With each need and ability a communication protocol – the *connector* – has to be declared. Currently different protocols are available providing communication via CORBA[3] structured events[4] for asynchronous communication via event channels, CORBA method calls for synchronous communication or shared memory in the case huge amounts of data like video information has to be transmitted. Other connector types like the streaming protocol RTSP[5] or

1 **T**ransmission **C**ontrol **P**rotocol / **I**nternet **P**rotocol
2 **L**ocal **A**rea **N**etwork
3 **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
4 Information about CORBA events and the OMG notification service can be found at "http://www.omg.org/technology/documents/formal/notification_service.htm"
5 **R**eal-**T**ime **S**treaming **P**rotocol

distributed shared memory have already been investigated and will likely be supported by DWARF in the future.

Each generic and reusable DWARF service extends the framework and is therefore of special interest to developers and system architects of future AR projects.

1.3.2. Middleware

Connecting services is the task of the DWARF *service managers*. Each computer which should be part of the Augmented Reality setup may run a service manager that handles service connections on this computer, to and from it. Considering all service managers on all integrated computing systems they represent the *distributed middleware* based on CORBA, that has been developed by the Object Management Group (OMG)[1]. To learn more about CORBA the reader is referred to [MAC01] (chapter 4.1), [HV99], [PRI99] and [SIE99].

The DWARF middleware is the negotiating layer which enables services to find and connect to each other spontaneously no matter if they are running on the same computer, on different machines in the room or somewhere in the Internet. The term middleware does not necessarily imply that there is a central negotiating system “in the middle of the ware”. Instead service managers which are distributed over the computer network support this task making network-transparency possible only now.

The distributed middleware also supports platform independence and portability of source code since the underlaying task for inter-process communication is handled by CORBA Object Request Brokers (ORB)[2]. For each platform and programming language of a service's implementation a suitable ORB can be chosen. Even for small handheld devices compact ORB libraries are available in the meantime.

To make use of the DWARF framework a service has to provide a valid service description and must register with a service manager either running on the same computer or a distant one. Note that it is not obligatory that a service manager runs on each computing system. Once registered the selected service manager searches for matching abilities for the service's need by additionally querying all other service managers which are found using the service location protocol (SLP). If another service can be located satisfying the need, both services are connected to each other. From now on they can communicate using the specified connector protocol without additional help of the service managers.

Collaborating generic DWARF services dynamically form basic AR systems. With the help of additional specific services they can be configured to the need of particular applications. This allows to build new AR applications rapidly without having to create the basic AR subsystems from scratch.

1 Web site of the Object Management Group: “http://www.omg.org”

2 A list of links to various ORB vendors can be found at “http://www.cetus-links.org/oo_object_request_brokers.html#oo_corba_orbs_ORBs”

1.3.3. System Requirements

In his thesis Tönnis explains that a conceptional AR architecture "[...] *defines the basic structure of Augmented Reality systems which can be constructed with it. Thus it ensures that service developers agree on the roles of their own services within the system and on interfaces between them"* [TOE03].

The addressed architecture covers the system tasks shown in figure 3. The stroke-dotted arrows represent dependencies in UML[1] notation.

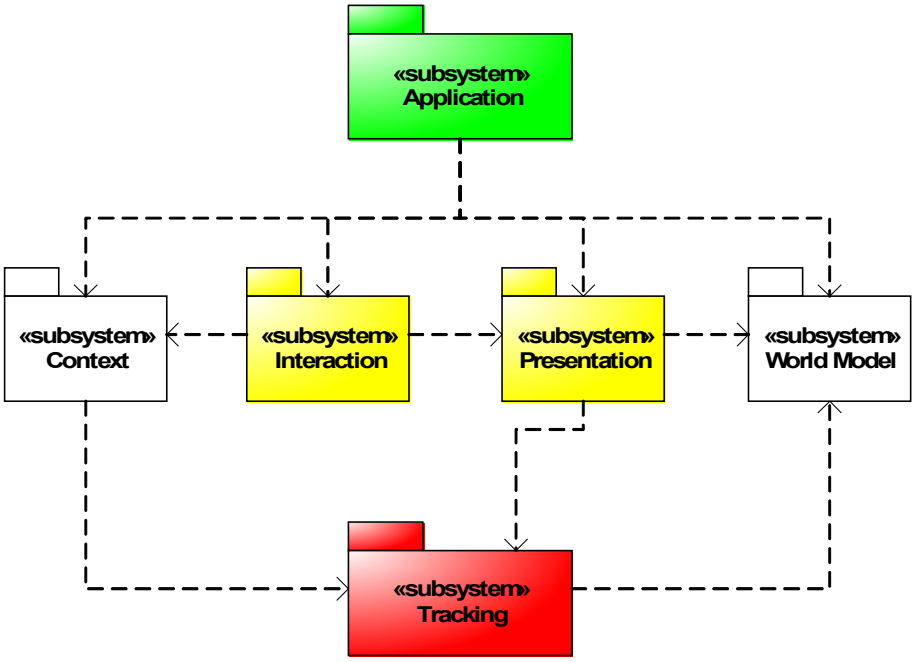


Figure 3: AR subsystem decomposition

The following sections explain these tasks, which are supported by Augmented Reality frameworks and thus also by DWARF.

Sensoring varying Information from the real World

This is the most important AR task handled by the *tracking subsystem* which consists of a composition of services in the context of DWARF. The key feature is the retrieval of the position and orientation (together six degrees of freedom) of users and objects in the real world like within a room, on the campus, at the place of work, on the road or elsewhere. This is necessary to determine the locations where the virtual objects have to be placed, and to apply perspective distortions to them in order to provide a more realistic impression. The acquisition of the user's location data has to be constant over time at a high update rate such making AR applications possible only now.

The tracking subsystem provides the connecting layer between the physical tracking devices

1 **U**nified **M**odeling **L**anguage

and the other DWARF services which are dependent on the location data. By the use of quality-of-service indications these services can find out how precise the tracked information is, and demand data from another available tracker if the accuracy is insufficient. As part of the tracking subsystem layer a *tracking manager* supports the generic access to the underlaying trackers which eases the integration of new sensor devices. There are a number of such devices available which I would like to categorize into two classes at this point.

Stationary Sensors Fixed to particular spots within the AR target environment these sensors provide information about persons or objects moving within range. These intelligent environments are often called “AR-ready” and usually offer dedicated computing resources which provide dynamic access to the tracking devices. When making Augmented Reality applications accessible to the public, stationary sensors will be employed rather rarely since the set-up for intelligent environments – when done on a large scale – is very expensive. As an example for a stationary tracking device the so-called “outside-in optical trackers” can be mentioned here. They represent optical cameras which are often mounted to the ceiling of rooms, and provide video streams from which the user's position can be extracted. For more information about retrieving 3D coordinates of objects by examining video images the reader is referred to [HZ00] chapters 5, 8 and 9.

Figure 4 shows the KidsRoom[1], an AR project developed by the MIT Media Laboratory Digital Life Consortium. This example demonstrates a setup of outside-in optical tracking systems. The KidsRoom is about an augmented children's room and therefore addresses children as its users. Their movements and actions are tracked by three[2] stationary cameras which are mounted to the wire-grid ceiling (see figure 5) 7,62 meters above the ground.

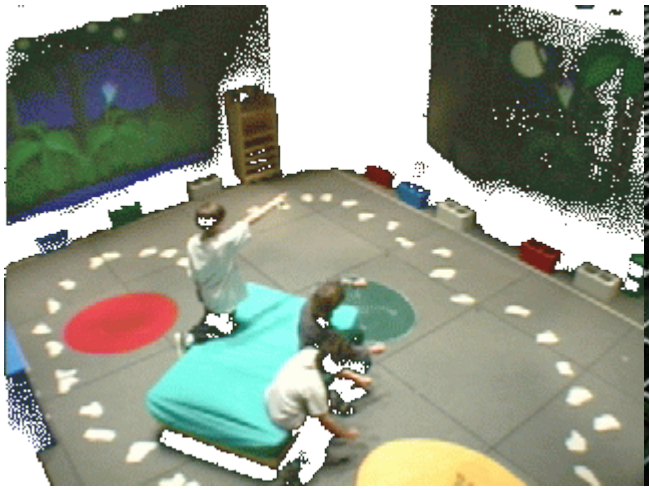


Figure 4: A spectator's view of the "KidsRoom" at the MIT Media Laboratory

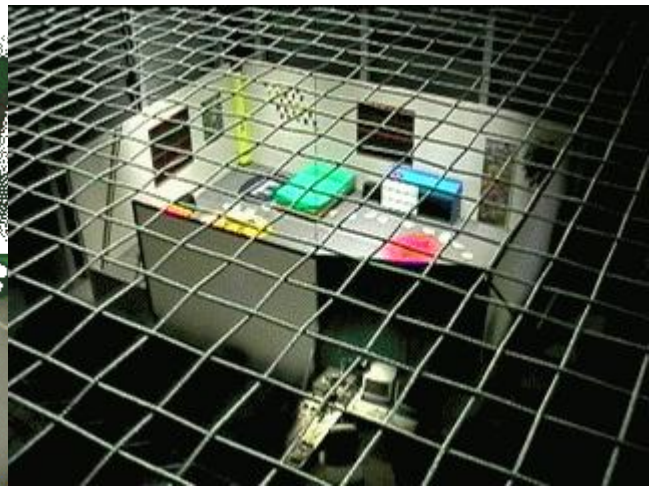


Figure 5: The wire-grid ceiling above the MIT "KidsRoom" where four stationary cameras are mounted

1 http://vismod.media.mit.edu/vismod/demos/kidsroom/kidsroom.html

2 Actually four cameras are used in the project, but the fourth one only captures video pictures for spectators, and is therefore not part of the tracking system.

Mobile Sensors By using portable sensor systems the user's environment either does not need to be modified at all or only has to be extended with some few additional supports. For example, devices like GPS[1], ultrasound or radar are often integrated in test vehicles when innovative AR applications are tried out by car manufacturers. Other approaches use inertial sensors or also hybrid trackers, where different systems are combined to increase the accuracy of the examined location data. All these addressed sensors do not have the necessity to add further information or any technical devices to the user's environment where they are employed.

In contrast to them “inside-out optical marker trackers” need additional information about the surroundings of the real world where the AR application will run. A head-mounted camera captures the video pictures of what the user is seeing. If the view contains a marker it can be detected and used to estimate the user's position and orientation. The prerequisite for marker tracking is that the markers have to be fixed somewhere in the environment so that the camera picture contains at least one marker all the time. Figure 6 shows an example marker of the ARToolkit[2] [KBP05] which is a software framework enabling the determination of the camera pose with respect to the tracked markers in real time.

Figure 6: Marker which can be detected by the ARToolKit

More information about marker detection in Augmented Reality applications can be found in [WIE04] and [ZFN02].

Combining Reality and Virtuality

As explained in the previous chapter tracking systems enable the access to dynamic and changing information within the real world. There is also a lot of static information an AR system must know, e.g. the arrangement of rooms in a building or the locations of streets in a country. If all these should be combined with synthetic data an information base is needed to merge both dimensions – the real world and the virtual world. In DWARF the *world model subsystem* encapsulates an efficient tree data structure – the scene graph - of real and synthetic objects using homogeneous coordinates. Each node is assigned a parent so that the child's position and orientation values are stored relatively to the parent's ones. When doing the rendering process this scene graph is traversed. The world model also acts like a “facade” as it provides access to all of the registered objects, which represent the augmented scenery.

Enabling User Interaction and Presentation

For supporting the AR requirement of real-time user interaction DWARF has the user interface engine as its disposal. Besides the tracking of physical objects as an AR input metaphor

1 **G**lobal **P**osition **S**ystem
2 http://www.artoolkit.org/

it supports multi-modal user inputs like the combination of voice and gesture recognition, for instance.

Since AR is about augmenting reality, the virtual information has to be made perceptible to the user. As seen before the presentation is generally done by means of three-dimensional views of virtual objects on head-mounted display or other display devices. Such an object could be a virtual model of a physical person who is not present, but with whom the user is currently talking over the Internet.

Besides the presentation of 3D structures the physical world can be augmented using sound or simply text information. During waiting for the next city bus the user can get various information by just looking at the bus stop sign through her head-mounted display. Text messages may then be presented fixed in relation to the sign and depict the time when the bus arrives, the route it will take or also news reports and the weather forecast, for example.

Supporting Applications Tasks

Application tasks can be configured as workflows, i.e. a series of distinct states where each state may conditionally be triggered depending on the previous step. These conditions are represented by the user's input or by events released via an environmental change. The configuration is declared to the taskflow engine by the system architect. At runtime the taskflow description is then converted to a finite-state automata representing the application's conditional workflow. Associated information like for example maintenance manuals which have to be presented to mechanics[1] at certain states in the repair process are linked in the taskflow configuration. By separating the application logic from the information content, mock-ups of new applications can already be created at an early design phase in which modifications are still easily done.

1.4. What Problem does this Thesis address exactly?

The thesis is aimed at providing a management software architecture, that helps AR applications to deal with spatial information of real and virtual objects. The information handled by the architecture is related to the users' situational environment, and describes the location and spatial relationships of users and surrounding objects in the augmented world.

Chapter 1.2 already introduced the reader to Augmented Reality and also to some representative example applications and their general properties. This section explains what can be understood by spatial information and points out its importance for AR applications.

Spatial Information Let us first take a step back from AR and consider an example from daily life to start with, before looking at a general definition of what spatial information is about. Imagine an unfortunately rather frequently occurring traffic situation – a rear-end crash.

You are breaking in front of a road intersection and another driver is crashing into your car

1 This example is related to the TRAMP project at the TU München which is briefly described in chapter 1.2.

from behind. Instead of apologizing to you for damaging your new car, the other driver blames you for the accident. Of course you point out that you had to regard the right of way at the crossroads in front. So you gradually slowed down your car. But the other driver blames you for unnecessarily stopping all of a sudden.

This argument contains a lot of spatial information such as positions, orientations, velocity and acceleration values (negative acceleration means breaking) with respect to different points in time before and when the accident happened. The initiator of the crash probably remembers the moment when awaking from the carelessness and pushing the break pedal. At that point in time this driver will certainly have noticed that the spatial information “stopping distance to car in front” represented a too large value to avoid a crash since the spatial information “safety distance to car in front” showed a very small value indeed.

Definition 1: Spatial Information

Spatial information *represents any information that is directly related to a location in space.*

Spatial Context Spatial information is not just limited to position and orientation data. It also comprises complex spatial relationships between users and objects depending on the situation in which they arise. Here the spatial properties of the users' “*context*” is addressed – a term that will be investigated in detail in the next chapter.

Supporting AR AR applications require to be informed about a lot of *spatial information* appearing in the user's environment. Besides the user's own spatial information this also includes rapidly changing data of other users, animals and objects which are concerned to be near to the user. This information is crucial when augmenting the user's view correctly. In addition the extents of the applications increase more and more thus also further increasing the quantity and variety of spatial data. For instance in the area of AR applications for cars, location data of lots of vehicles have to be acquired, evaluated and placed at the application's disposal.

Since it is difficult to describe all possible reactions of the various implementations to certain constellations of spatial data it is desirable to have a generic software system which abstracts from common requirements and manages the acquired information in dependence on a configuration of the application domain. If a user is piloting an airplane an AR application supporting this scenario will have other properties and requirements to the management of spatial data than a setup for an augmented city guide where the user is mainly walking on foot. In addition to basic spatial data it is also possible to deduce additional information like the passing time until an object collides with another one.

For the presented system suitable approaches and methods are investigated which regard this information management. Set up on this knowledge a spatial context management frame-

work is presented. This framework is intended for the utilization in distributed applications, and enables the management of spatial context regarding the user's situational environment.

Overview of intended Functionality and Requirements Since it is possible to find general characteristics of spatial information a common information pool could be used describing the general relationships between the various entities to which the information belong. When creating new AR applications, system designers can fall back on the information source and extend it to their needs thus increasing the strength of the framework. The system should provide a uniform querying mechanism which does not constraint its flexibility if possible but ease its usage. Therefore also a distinction between the continuous acquisition of spatial context data and its reusable employment can be achieved. The framework should additionally provide a mechanism for supporting the task of interpreting the aggregated spatial information on the basis of a configuration describing the interpretation procedures.

Target Environment DWARF is frequently used at the Technische Universität München for building AR applications. For testing and demonstrating the new system, this AR framework is used. However the system should not be dependent on it, so that other frameworks can also be addressed.

2. Context Management

Augmented Reality systems have to deal a lot with contextual information as explained in the previous chapters. There the expression “situational information” has been used in place of the term “context” to give the reader the basic background knowledge for the explained topics.

This chapter is aimed at defining this yet rough idea of context and context-awareness. Furthermore context-aware applications are investigated and the requirements they must satisfy to handle context are discussed. Since the presented system addresses spatial context which is a subset of general context, this will be furthermore investigated when discussing the requirements for supporting the spatial context management for applications in Augmented Reality. The succeeding section finally explains different approaches of representing spatial context in software systems.

2.1. What does “Context” mean?

When talking to other people, and stating the term “*context*” in the conversation, people often intuitively now what it means. But if one asks somebody to define it, she is likely not able to do so. The reason for this are the various meanings of this word depending on the context, the term “context” is used in.

Before any more confusion let us have a look at figure 7 [FIS01]. Afterwards definitions with special interest regarding computer science are provided.

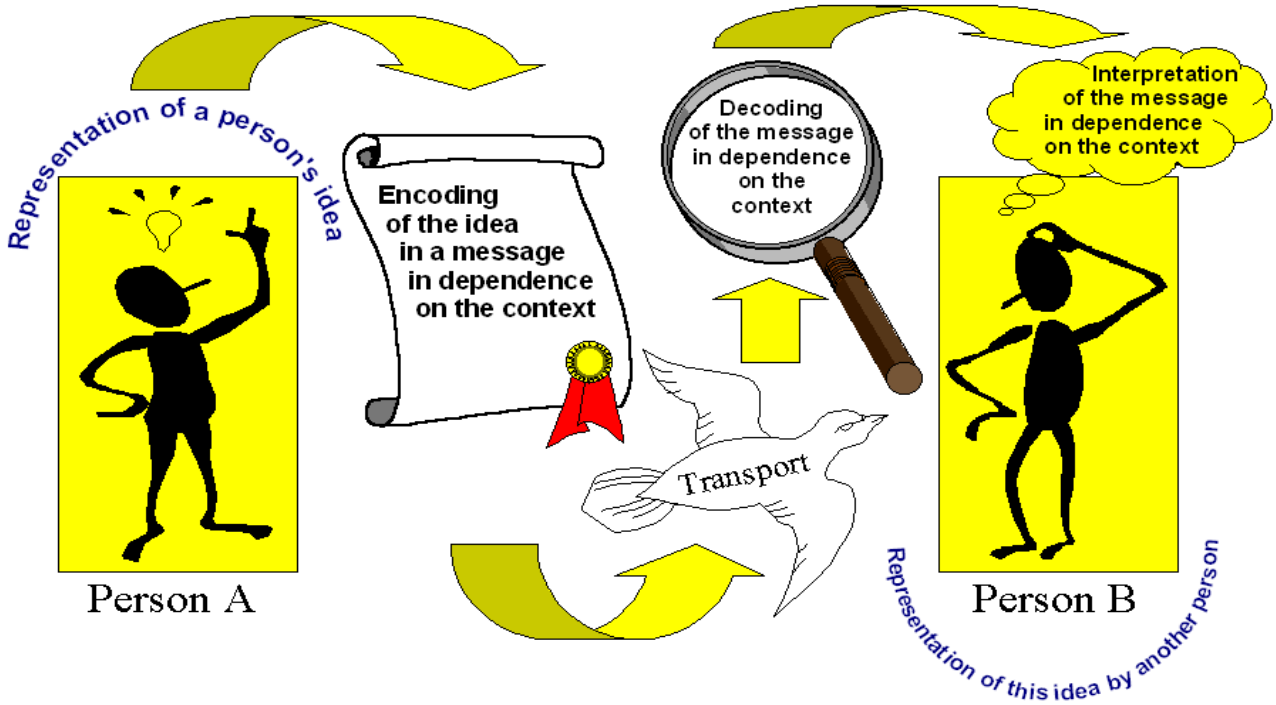


Figure 7: Context-aware communication between two persons

By using speech, pictures or notes people are encoding the information to be sent, and transmit it with the help of a suitable medium. When communicating directly with a faced person people can talk, whisper, shout, sing, cry or hand over the necessary information. Other mediums are for instance telephones, letters, electronic mail, news letters, web sites, web logs or IRC[1]. Also video messaging, television and radio or the mobile phones' short message services can be used as a medium for transmitting notices to one ore more other persons. This is done by implicitly including the sender's situational information – the context.

Normally the transport procedure does not modify the message content that is finally received and decoded by the one ore more addressed persons. The receiver does not know the exact context of the sender, and additionally interprets the message by referring to the own context. If both contexts differ too much, the statements of the message is often not very useful. People realize this fact when sending messages to wrong persons by mistake.

For example, when sending the electronic message “When I am finishing work soon, perhaps I make it for dinner in time.” to a colleague instead of the partner, the receiver would certainly have difficulties in interpreting the original information correctly. On the other hand the quality of interpretation increases if both contexts overlap. Above all the composition of different communication media contributes to a robust interpretation. Creating some additional sketches on a blackboard during simultaneously explaining theoretic study content eases the understanding and interpretation of the information which depends on the teacher's context. Besides the increase in the accuracy of information exchange there are other very important reasons why context is used. By involving context in our communication the information throughput is increased and the amount of the necessary transmitted data is decreased. That refers to the core of the statement of two persons who know each other for a long time and tell that they understand each other without saying much.

Many definitions of context are trying to paraphrase this term like “aspects of the current situation” [HNB97] or “elements of the user’s environment that the user’s computer knows about” [BRO96]. In 1994 Schilit and Theimer [ST94] defined context as the location and identity of persons and objects which are near to the user. This definition also includes the constantly changing situation of the user's social, physical and also computing environment over time. In computer science the common problem of these approaches is the attempt to declare a specific always matching definition which holds for all kinds of situations. For example, if the temperature measured in a room is not important to an application at all then it does not contribute to the user's context. On the other hand there might be an application which has to evaluate this physical context so that the definition has to be extended again. Since it is not possible to list all situations users might experience, a more abstract definition of context is required.

Dey and Abowd provide a definition of “context” in [DA99] which is met with worldwide researchers' approval.

1 **I**nternet **R**elay **C**hat

Definition 2: Context

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.*

A situation is defined by a configuration of a number of states. Applications can evaluate the configuration and deduce a changing situation if the information values, that regard a certain aspect of the user's context, cross common thresholds. This implies a modification of the entities' environment and is used to activate certain routines of context-aware applications that are described in the next section soon.

It is possible to distinguish two different levels of context where the transition between both are ambiguous however. In spite of it the following two passages are trying to apply a qualified separation.

Low-Level Context This level of context describes the physical environment of an entity and represents all its contextual information which is directly retrieved from sensor systems. An important information is *time* because all other categories of context are dependent on its value. Examples for low-level spatial context are position and orientation, the entities' dimensions, weights, velocity and acceleration into different directions. In an application scenario where vehicles are involved a lot of additional physical quantities are included. To mention only a few examples vehicles provide the stopping distance and breaking time, multiple engine parameters, various kinds of measurable angles like the steering angle, the angle between the turned front wheels and the vehicle's longitudinal axis and many more. In addition low-level context relationships are also part of this context category.

Examples are the passing time until two entities are meeting each other or when staying with the vehicle scenario the way needed for overtaking another car can be listed here. As one can see there are innumerable context aspects which make up only the physical environment. In addition to these there are environmental conditions like noise, temperature, atmospheric pressure and humidity, brightness and air pollution. Also the computer environment contributes to the low-level context of entities. The infrastructure of computing systems lists the surrounding devices and software systems, their capabilities, availability and access control. Also the system's state, its internal configuration and system events are part of the low-level context.

High-Level Context Situational aspects of high-level context are related to the behavior of the involved entities. In principal they refer to human factors as the entities' habits, their emotional, mental and physiological conditions and the social situation with respect to their relationships within the environment.

For example, the state of the user's health, hers mood, the nearness of other people and the way other persons influence the social situation regards high-level context. It is rather difficult

to develop applications which have to evaluate this context category. It is possible to deduce higher level context from various low-level situational information. An application which is employed in a user's car could conclude from a hectic way of driving and the knowledge that the driver is late for a meeting, that she gets upset about her being late and therefore is in a bad temper. To help her calming down the application would switch the radio station to one playing classical music and would adapt the inside temperature to a comfortable degree.

The kinds of context are not only of great variety but also have an interesting recursive property. The context occurring at a particular moment is dependent on the context before, and also includes it. When driving with your car on the motorway their would have been a context change if you had a flat shortly before, and your car now runs on the spare tire which is restricted to a maximum speed of 80 kilometers per hour. Without the flat your context-aware velocity assistance system would indicate a recommended speed of 130. But because of the previous context information it should instead display a warning if you exceed 80 km/h.

This section already introduced some examples of context-aware applications. These represent the topic of the succeeding chapter where also a formal definition of context-awareness is provided to the reader.

2.2. Context-Awareness and Context-Aware Systems in Ubicomp

Software systems which are aware of the situation and surroundings require to access computing resources and services in order to retrieve contextual information and interact with their environment.

In everyday life users should not be concerned with the details of configuration or connection procedures of these services, which may be sensor systems or also nearby print servers, for example. Independent where the users are located, they rather should not even notice all the necessary computer equipment embedded in their environment. This new approach, which is aimed at replacing the traditional desktop PC paradigm by invisible seamlessly integrated computing systems into the physical world, has been introduced by Weiser in 1988, who called it “*ubiquitous computing*” or “*ubicomp*” in the abbreviated form [WEI91].

As stated in [GAL03] “*In its broadest sense, ubiquitous computing is currently seen to comprise any number of mobile, wearable, distributed and context-aware computing applications.*” Instead of compelling people to adapt to computers, computing services and their interfaces, these should adjust to their users to be usable in a natural way anywhere and anytime. As a matter of fact ubiquitous computing requires new forms of user interfaces, which should give the users a natural way of interaction with computer systems. That means that users should be able to use ubiquitous computing just as all other objects which are physically located in their environment.

Ubicomp involves heterogeneity on a large scale [SBWL04]. The heterogeneity of computer systems not only covers hardware and software platforms, networks and protocols, programming languages, middlewares and media formats, but also applications which may migrate to

other devices and have to adapt to the new context. Up to now often the different systems are not event meant to inter-operate with each other, let alone the dynamic discovery of each others' provided services and their usage.

Ubiquitous computing is directly related to *pervasive computing*, which also addresses intelligent or smart environments in which large numbers of computer systems are more and more mobile and globally available. Though both terms are used equally in general, Mattern explains, that the difference lies in their utilization. According to him ubiquitous computing regards the long-term academical appliance for idealistic visions, whereas pervasive computing refers to the short-term commercial usage in electronic-commerce and Web-based business processes [MAT01].

Context-aware ubiquitous computing integrates and joins two approaches. On the one hand ubicomp is aimed at making computing globally available. On the other hand the sensitivity to context requires sensing and interpreting local environmental situations of mobile users by *context-aware systems*.

2.2.1. Context-Awareness

To enable applications to be aware of the users' environments the systems are dependent on information provided by hardware sensors. These are becoming inexpensive over time and are not only employed in limited ranged test environments but also more and more frequently in daily life.

For example, in London video surveillance is installed at almost every crossroads in order to deterring from and fighting against crime. The increasing amount of information which is provided by these cameras can not be handled by the officers on their own. Image understanding context-aware systems can be used to cope with the information overflow. The task of evaluating and interpreting sensor data can be transfered to these systems to a certain degree. As another example RFID[1] tags will soon replace the bar codes which are attached to products in shopping centers. These passive tags can be read by scanners if they are in range. The utilization of customer data allows various new services. If a customer puts noodles into her shopping cart the aware system might suggest a proper sauce, based on the information which is acquired from the RFID tag of the noodles.

But the new technology does not only involve advantages. Commissioners of data privacy protection are afraid of the misuse of accumulated personal data which allows the development of profiles regarding the visited locations and the personal preferences of particular persons. Among other requirements also this topic, that is of particular interest when dealing with *context* and *context-aware applications*, is discussed later.

In principal context-aware applications are aimed at supporting the human-computer interaction by deducing additional input information from the user's context. Figure 8 depicts the well-known traditional human-computer interface within the yellow box. This user interaction is

1 **R**adio **F**requency **Id**entification, http://www.rfid-handbook.de/

called an explicit sequential input/output system. By analyzing the user's context input an application now becomes "*context-aware*".

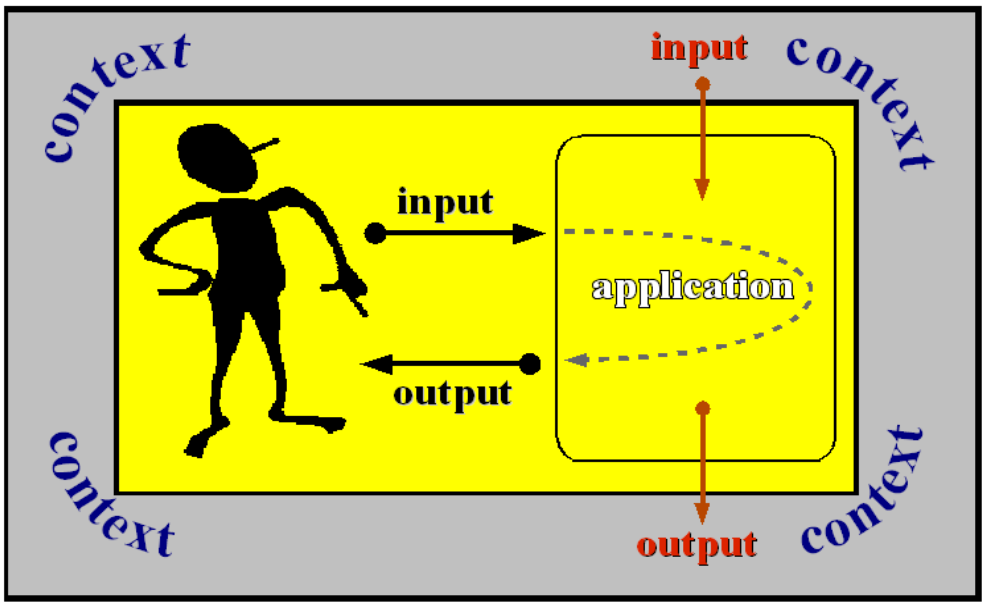


Figure 8: A context-aware human-computer interface.

This additional context input depicted as the surrounding area of the yellow box can be acquired from various environmental sensors such as location, velocity or also temperature sensor systems.

If the user issues the command "Increase the temperature!" then a context-aware home environment application would raise it in only small steps since it knows the user's preferences and also recognizes that the temperature is already above 24 degrees centigrade. In contrast it would increase it directly to e.g. 20 degrees if it was 10 before. When processing contextual information the context-aware system also contributes to the context which applies to the new situation after that.

But context-aware applications do not just adopt themselves to changing context which they are informed about as described by Schilit and Theimer [ST94]. Indeed their definition is only suitable for context-sensitive systems which do not contribute to a changing situation. The awareness of context mainly involves three questions which are continuously issued by the context-processing application and represent the basic aspects of context-awareness. These questions have first been stated by Schilit, Adams and Want [SAW94] in 1994 and are listed in slight modification as follows.

- Who is my user?
- Which entities are near to my user?
- Which resources are in reach?

In addition to these issues context-aware applications also have to embed the user's activity

and address its level of importance. These demands to context-awareness of computing systems have been defined by Dey and Abowd in 1999 [DA99].

> **Definition 3: Context-Awareness**
>
> *A system is **context-aware** if it uses **context** to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

Therefore an application has to be conscious of the user's context and tasks. It must be able to acquire the contextual data, evaluate these, adapt its own state of context-management and use the gained information to help and support the user to perform a specified task. In general, context-aware applications are looking for the **who**, **where**, **when** and **what** of entities to conclude **why** a situation takes place.

But why should context-aware applications be developed and used? On the one hand mechanisms for context acquisition, aggregation and interpretation have to be investigated and implemented in order to enable context-awareness at all. On the other hand context-aware systems are able to perceive and interpret their environment which sets them into a position to react appropriately to the situation. The system should detect and understand the intentions of its users and should often be able to automatically take actions on behalf of them. User input can mostly be deduced from the context which increases bandwidth and fault-tolerance of the human-computer interface. Consider the case where a user is missing some information which have to be input explicitly. If the user's context was estimated by the system, the application could possibly complete the unavailable data. This is most convenient for users who are new to computing systems. But it also increases the information input by practiced users because they do not have to specify all data explicitly.

2.2.2. Categories of Context-Aware Applications

There is a great variety of context-aware applications. This section provides a classification by means of common categories. In 1994 Schilit enclosed four different categories of context-aware applications [SAW94] that are listed in the following table and are presented briefly.

Context-Aware Application...	**by manual initiation**	**by automatic initiation**
retrieves information for the user	**Proximate Selection**	**Automatic Contextual Reconfiguration**
executes a command on behalf of the user	**Contextual Commands**	**Context-triggered Actions**

Table 1: Four categories of context-aware applications

Proximate Selection This user-interface technique should ease the selection of nearby services (e.g. wireless LAN access points or printers) and information about people and objects in a specified range around the user. This is done by emphasizing nearby results of the manually initiated query. For instance, one could request the system to show all friends which are currently at the campus. Then a list of these is displayed which is sorted by the distance or other properties. Perhaps the system provides an appropriate link to a messaging service with which a notice can be sent to the selected friend.

Automatic Contextual Reconfiguration Like *proximate selection* this category of context-aware applications provides information and services to the user. But in contrast to the previous definition applications are not dependent on explicit user input. They evaluate changes in the user's context and reconfigure their behavior accordingly. In Augmented Reality this technique is often employed when trying to create the impression of physical objects which are only virtual. Also the automatic connectivity of DWARF services can be seen as an example. When new services are registered with the DWARF service manager the whole system is automatically reconfiguring itself to the need of these services.

Contextual Commands When the user presses the print button the context-aware application automatically uses the nearest available printer to execute the print-job. This simple example illustrates the approach of contextual commands. Since the initial definition by Schilit people often unintentionally mix it with the *automatic contextual reconfiguration*. The important difference is that the user is required to manually initiate the application's context-aware function which analyzes the user's context and automatically executes a matching command. Because people have a lot of habits they often find it frustrating to tell e.g. the coffee machine that it should put some extra milk and no sugar into the coffee. An integrated application using *contextual commands* would automatically prepare the proper coffee for each user after she pressed only one button or inserted the correct amount of money.

Context-triggered Actions Applications regarding this category are configured by pre-defined rules. They continuously analyze the rule conditions and automatically execute the configured commands if a rule holds for the current context. For example, when going on holiday an application could be configured to fill the cat's feeding bowl at most three times a day and only if the animal is in the room.

2.3. Requirements for the Management of contextual Information

As explained before in chapter 2.1 when discussing the properties of low-level context, all contextual information of entities include their location. It is obviously hard to separate the concerns of general and *spatial context management*. A distinction can be given by means of the kind and complexity of the representation of entities' location-based information and their

spatial relationships. Before further going into this topic in detail this chapter is aimed at presenting the general requirements, which have to be met by a software system, that handles context information.

This chapter introduces to the prerequisites, which have to be met by a context management system in order to support its functionality. Because spatial context management shares the aspects of the general handling of contextual information, the following section first describes the common requirements for context management whereas the succeeding sections concentrate on the aspect of spatial context and Augmented Reality. The last chapter finally presents the qualities context management system must satisfy in order to provide the functionality at all.

2.3.1. Basic Functionality for Context Management

In the section 2.2.2 context-aware applications have been categorized. But an important distinction is still missing. Does an application represent a specialized implementation for one particular problem, or does it set up on an underlaying generic context-management system? The latter one will be concentrated on because specialized systems barely support any modifications and do not provide the necessary flexibility which is needed when dealing with diverse situational data.

Working with context information is not possible without a common mechanism or procedure that defines the way context is addressed and selected. Similar to database management systems some kind of “primary key” is required which should be used to index context items.

Primary Context versus Secondary Context In combination with aware systems context is characterized by distinct properties which represent the *primary context*.

- the identity of the entities which are part of the context including the applications' users
- the location of entities which are part of the context
- the time where this context was acquired by the system
- the application the context is managed for

It should be evident that the primary keys of context items can be used as indexes to find and address all other contextual information, the so-called *secondary context*.

In general, context management systems are distributed and have to satisfy common requirements. An important aim for supporting the facilitated access to and sharing of contextual information between separate applications is the so-called *separation of concerns* between the continuous acquisition of context and its usage by aware applications [GEL01].

The general requirements, which have to be addressed by (spatial) context management system, are briefly listed in [DSA01], and explained in detail as follows.

2.3.1.1. Context Acquisition

There are numerous characterizations of contextual information. Since in the most cases it is very difficult to directly retrieve higher-level contextual sensor data people often have to be satisfied with basic information about the physical environment. For example, when analyzing the user's current mood one could invent a “mood-O-meter” sensor which tells whether the user laughs, shouts or also cries. Certainly the same task is achieved by only acquiring basic data such as the tone of voice, the frequency of the spoken words and the noise level, and analyzing their relationships afterwards.

When thinking of multi-modal input devices in Augmented Reality, also context-aware applications are often dependent on multiple heterogeneous sensors. These may provide explicit user inputs and/or implicit contextual inputs which that describe related information. Such arising redundancies increase the information accuracy when they are combined and interpreted properly.

Time is a constituent of primary context, which is used for indexing newly acquired situational information and historical context. As a matter of fact a global time clock mechanism is necessary to guarantee, that all managed context is consistent with respect to its chronology.

2.3.1.2. Aggregation and Access Mediation

The acquisition of contextual data is faced with aware applications requiring to access them continuously. The *aggregation and access mediation* has the task to collect and structure the context data provided by the sensor services. The more different kinds of contextual data is aggregated the more situational information is “condensed”. Because each data set contributes to the description of a particular entity's situation, usually the information aggregation is done with respect to this entity. Therefore it is possible to distinguish areas of aggregation by means of the referenced entity.

The mediating part of the aggregation system encapsulates the uniform context access. In general it supports the “publisher/subscriber” or “push” pattern where context providers add information to the aggregation system as publishers and context receivers are then informed about the new data by the mediating system. In contrast to it the “pull” pattern enables a context receiver to directly query the information, that is held by a service of interest.

By strictly distinguishing between the acquisition and aggregation of context and its usage, it is possible to abstract a context interface similar to a user interface where the user's input is separated from its effects. Here this method enables the reusability of context by multiple applications and decreases the time required for developing new context-aware systems.

2.3.1.3. Reasoning

This term is frequently used in conjunction with the interpretation of contextual information. It stands for the logical inference with respect to the processing of knowledge, and represents the procedure of deducing higher-level context from low-level information.

For instance, consider a person exiting the video store with some DVD[1] films in the hands. An application could infer:

- The person likes movies.
- The person owns a device which plays DVDs like a DVD player in conjunction with a TV set or a notebook with a DVD drive, or knows somebody who owns such a device.
- The lending fee is two Euros per DVD for each day.
- The person plans to spend a pleasant evening watching the Star Wars Trilogy.

With the help of rules or also via automated machine learning the reasoning layer infers about *explicit* context that comes from the acquisition and/or aggregation system, and deduces *implicit* context information which was unknown to the system so far. The obtained results should be reusable, for instance as a basis for additional interpretation.

2.3.1.4. Detecting Inconsistencies and Resolving Ambiguities

Especially in distributed context management environments *data and logical inconsistencies* occur quite frequently. Data inconsistencies arise if multiple system components manipulate the same contextual information in different local copies without updating each other or mutually synchronizing using a global context representation. By using the experience of the transaction concept's properties ACID[2] in database management systems [KE01], it could be assumed that overall consistency is automatically guaranteed, so that inconsistencies do not require to be resolved at all. However inconsistencies also involves another cause. When incorporating a lot of context sources that provide related information about the user or her environment, the aggregated context may become logically inconsistent.

For example, let us assume that two sensor systems monitor the presence of persons, sensor "S1" in room "R1" and sensor "S2" in room "R2". As figure 9 shows, the aware system's context information is based on the knowledge, that a particular user is located in room "R1". Supposed "S1" does not sense for some reason, that this user leaves the room "R1", but "S2" detects hers presence as the user enters "R2", the context-aware system now contains the logical inconsistency, that the user is located in two different rooms at the same time. This in-

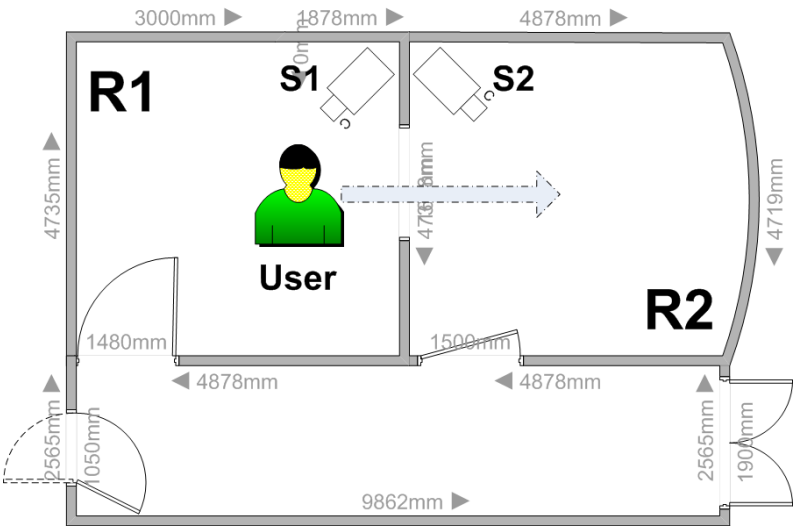


Figure 9: Example for the evolving of logical inconsistencies

1 **D**igital **V**ersatile **D**isc

2 The ACID paradigm consists of the properties **a**tomicity, **c**onsistency, **i**solation and **d**urability.

consistency could be detected e.g. by a logical inference about the conceptional knowledge, that individual entities can not be simultaneously located at two different places, and resolved by comparing the timestamps of both location information of the user.

These uncertainties on context information which refer to their multiple possible interpretations are called *contextual ambiguities*. Reasoning higher-level context from *ambiguous sensor data* implies faulty interpretation information. Therefore a mechanism should be provided which detects and resolves contextual ambiguities and controls access to it. It is often possible to improve the accuracy of the faulty data by comparing it with related information acquired earlier.

Another approach is the forwarding of the responsibility for resolving the encountered ambiguity [MAN01]. When the aware system is faced with more than one possibilities of interpreting the implicit or explicit input then it simply asks the user how to cope with the uncertain information. Services trying to retrieve ambiguous data from the aggregated context sources should either be postponed or warned about this property. After the ambiguity has been resolved notifications must be sent to all services that tried to access the data so far. Indeed it is good practice to also inform the user about this problem in any case, since ambiguities can impair security, privacy and sometimes even safety.

2.3.1.5. Persistence

Context is also depending on *historical information* describing the user's situation within a past period of time. Therefore mechanisms must be provided which enable the storage and later retrieval of context data. This is of great importance to the reasoning layer that frequently requires the context's history. Also applications are able to use situational input which was acquired before they have been started. A still common and open problem is the question how to deal with ambiguous context, whether to just discard it, store it together with unambiguous information or update it after the ambiguity is resolved.

2.3.1.6. Distributed, transparent Communication and Resource Discovery

Although there are approaches to centralized context management systems, they are always based on distributed sensor systems. Personally I think that the most promising projects will have a *spatially distributed communication infrastructure* at theirs' disposal, that hides the low-level details of network protocols from the communicating system components.

Contextual information is acquired from various sensor systems, whose connections to the aware-system are frequently changed as users move in smart environments. The context management system has to provide a *resource discovery* mechanism which allows for dynamically locating of and connecting to all resources, that are related to the context management in the framework.

Advantages of a distributed infrastructure supporting resource discovery are the raised availability of software components, the spreading of the computational load onto different hard-

ware platforms and the easy integration of systems that are discovered dynamically though they may be located far away. Also several applications can make use of distributed sensor systems and reuse generically implemented components.

In contrast the main disadvantages are the communication overhead and the increasing costs for resolving data inconsistencies which often appear when applying mirroring or replication strategies for contextual information sources. Dependent on the particular architecture design of the communication infrastructure changes to it may interfere with already existing applications, if the coupling of the communication mechanism is not implemented loosely.

2.3.1.7. Safety, Security and Privacy

Special interest should be addressed to the user's safety when she uses the context-aware system, and the *security* and *privacy* of sensitive context information. What happens if the system does not interpret the user's context correctly or if the sensor data is faulty? What if the driving assistance application, that is known from chapter 1.1, initiates an emergency break of the user's car just because the front radar sensor receives a particularly powerful reflection signal, and misinterprets a thrown away cola can as an over-sized truck blocking the road?

Besides safety, also security and privacy issues are of special importance. Context-aware computing is essentially based on the user's location information. Certainly this data can be misused to create profiles of the user's motions and the locations she visits. Security of sensitive data and privacy protection or rather a compromise between user's privacy and the applications' requirements has to be guaranteed in context-aware and ubiquitous computing sooner or later to establish the acceptance of these systems by their users. Another underestimated concern is the ordering about of the user and the loss of privacy due to the numerous necessary sensors fixed within the environment.

Security policies are the identification and authentication of users, the authorization and access control, the inference guideline, the reliability and auditing [FK02]. To be granted access to the application or context management system the user first has to identify herself to the system. Generally, this can be done by providing a user name. The system will then ask the user to authenticate herself by means of stating a password. If authenticated, the system authorizes the user to access certain context information depending on the kind of access rules. These rules map passive information to active accessing entities like users, groups or applications. The authorization procedure guarantees the secure direct access to information according to the user's privileges and the information's guiding rules of access control.

Auditing is a mechanism used for “taking notes” of access procedures. The created profiles are required to guarantee the reliability and the integrity of the contextual data. Intruders and their assaults to the system can be detected based on these profiles at an early stage, and the logical consistency of the managed context can be preserved.

2.3.2. Abstraction of Spatial Queries

In addition to the functionality discussed in the previous section, software systems, which are specialized in the management of *spatial context* must allow for *abstracted* spatial queries. As a matter of fact aware-application are released from stating low-level queries like they are required when accessing geographical information systems [LGMR99] or database management systems.

Examples for abstracted spatial queries are "*does the airplane approach the runway?*" or "*give me all entities on the street in front of the user's car, which are an obstacle for the driver*".

Frequently queries are range-based or represent nearest neighbor selections. Range-based queries expect a number of entities in a spatial area to be returned (e.g. "*select all taxi cubs, which are in range of Harry*") whereas nearest neighbor selections query for one ore more objects located closest to a given entity (e.g. "*retrieve the nearest bus stop*").

Abstracted spatial queries are dependent on the spatial context management system's interpretation of the entities' location-based information by analyzing their properties and relationships on the foundation of classifications and geometric or topological selections.

Classification-based Selections can be compared to the queries with regard to database management systems. They are based on the properties of the managed entities. Exemplary selections are "*select all vehicles which have more than two wheels*" or also "*give me all persons who generally use public transportation*".

Geometric Selections are processed with the help of basic geometric forms (e.g. point, line, area, solid), which locate and represent objects. They support selections for the geometric position of entities, the distance between points or also the size of areas (e.g. the spreading of a flood).

Topological Selections take advantage of the arrangement of objects in space without considering their geometric forms or expansions. Each object consists of an interior, boundary and exterior, and can share in topological relationships such as "disjoint", "overlap", "contains", "equal", etc. (see figure 10).

An example for a selection by means of the inclusion of objects in distinct areas is: "*select the entities which are located inside the house*". Though this topic will be further investigated when talking about representing spatial information in a later chapter, the interested reader is referred to [EH91] and [EM92] for more detailed information about topological relationships and a possible classification (the "9-Intersection Scheme").

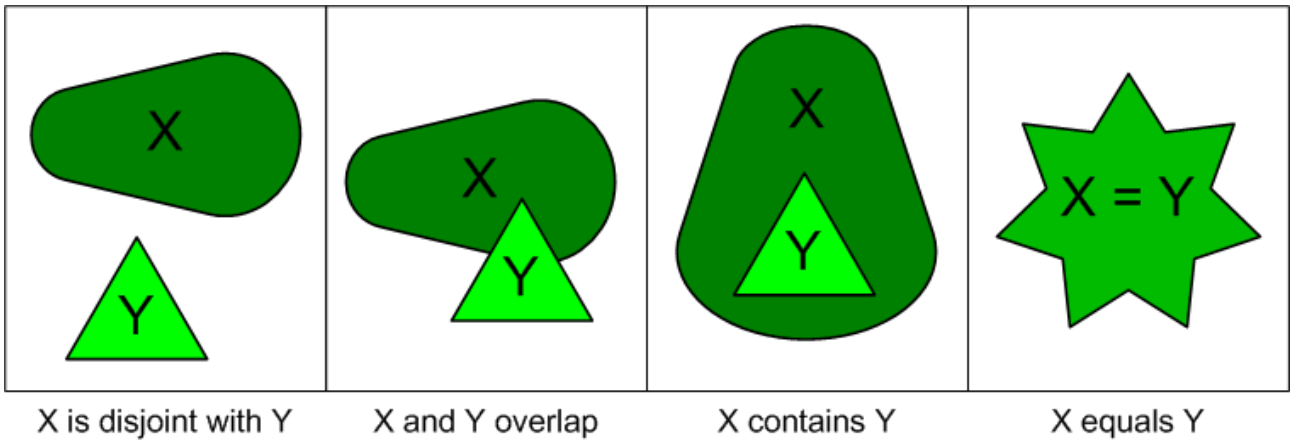


Figure 10: Exemplary topological relations between the objects X and Y

2.3.3. Functionality for supporting AR applications

According to the characteristics of Augmented Reality (see page 14), a context management system serving this domain must allow for physical as well as virtual entities. The additional functionality covers the acquisition of spatial context in the virtual world, which is seamlessly integrated into the aggregation process. An important requirement is the transparency with respect to the interpretation step and the processing of abstracted spatial queries regarding objects in the augmented reality. In this case the context of physical objects must be comparable to context of virtual objects and vice versa. This not only includes the entities' properties, but also their spatial relationships, which go beyond the border of both worlds.

Also for the domain of AR the disintegration of contextual inconsistencies is of special importance. For instance the placement of virtual objects or the attachment of virtual information to physical objects may be inaccurate. Here it is necessary to resolve the possible arising ambiguities on behalf of the supported applications.

In addition the persistence and the consideration of both the security and privacy aspects have to be extended to be also suitable for virtual entities.

2.3.4. Prerequisites for supporting Functionality

Several prerequisites are required to support the efficient execution of context managing tasks. These prerequisites are sometimes called “quality constraints” [OFF00], “nonfunctional requirements” [BD00] or also “quality attributes” [WHI05]. They are related orthogonally to the functionality of the context handling systems, and are founded on the individual context management architecture's design and implementation.

Accuracy of Sensed Data Context management systems rely on the accuracy of the provided information. Though their functionality include mechanisms for dealing with inaccurate data, these are only applicable to a certain degree. Therefore context providers must offer information about their quality of service.

Fault Tolerance A context-aware system has to provide a sufficient extent of fault-tolerance. If for example one of its modules or a sensor system fails, it must be possible to dynamically reconfigure the setup and integrate compensating components to recover the required functionality.

Performance In order to allow for the real-time constraint of AR, performance-related requirements such as the throughput of information, the response time to abstracted spatial queries and the low latency for the communication in the distributed environment are especially hard. Certainly it is not easy to achieve these qualities in view of unstable communication rates between and rather low hardware resources of mobile devices, which are involved due to the mobility aspect.

Mobility also refers to the AR domain. The system must support *wearable* and *mobile* computer devices such as HMDs, handheld computers and car navigation systems.

Scalability refers to the dynamically changing number of software and hardware components in the environment of the users. It particularly includes the dynamic changing number of applications using the system, the amount of incorporated sensor and information systems, the number of managed entities as well as the changing complexity of context abstractions.

Interoperability of soft- and hardware systems aims to bringing together what is not necessarily designed for working together. Here the fundamental problem is heterogeneity with respect to operating systems, communication mechanisms, programming languages and software interfaces. Therefore software abstractions, bridges and loosely coupling of components are required for supporting ubiquity.

Location-based Selection is required for dynamically adapting the computing infrastructure to the location of moving users. The system must support means for selecting nearby software and also hardware resources like printers or video beamers. Because of the multiplicity of these resources (i.e. they are reused by other systems), it must be possible to agree on sharing them.

2.4. Spatial Context Representation and Modeling

This chapter introduces the reader to different approaches to represent and model selected location-based properties of reality. First conventional spatial context representations are discussed. Besides their usage in generic context management frameworks these models are often employed by specialized applications that manage spatial information on their own by means of local context models. These context models and an appropriate classification are

presented in chapter 2.4.1. Another approach in spatial context representation and modeling is represented by contextual ontologies that are based on research efforts with respect to the Semantic Web. Contextual ontologies are explained in chapter 2.4.2. The benefits and limitations of both specific approaches are rated at the end of each chapter. Finally the combination of context models and contextual ontologies is discussed in chapter 2.4.3.

2.4.1. Conventional Spatial Context Models

Committing the representation of contextual information to virtual models impairs the conceptual design of aware systems. As described previously an important claim of context management systems is the sharing of knowledge between separate applications. Therefore a resulting demand is the integration of a common spatial context model. According to [BN04] figure 11 depicts the employment of a context model for supporting multiple applications.

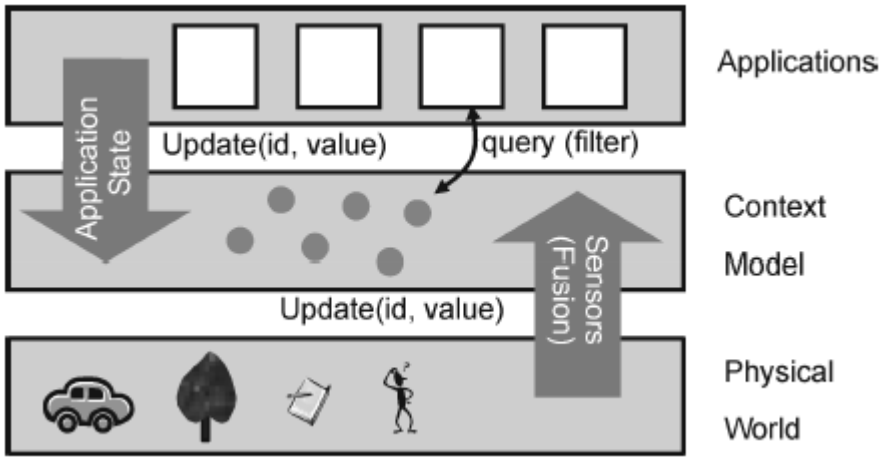


Figure 11: Generic spatial context model

The model acts as a mediating layer between the acquisition of sensor data of physical entities and the usage of aggregated spatial context by different applications. By accessing and updating a common knowledge resource, sharing of contextual information is enabled.

Spatial context models are data structures acting similar to database management systems. Although their internal organization may differ with respect to the model's instance, they have to meet the common requirement of providing access to structured data. As a matter of fact a resulting advantage is the simplified integration of a spatial database management system like IBM DB2 Spatial Extender[1] or Oracle Spatial and Oracle Locator[2]. These provide dedicated mechanisms for storing and retrieving spatial information of entities.

1 IBM DB2 Spatial Extender represents a software extension to the IBM DB2 Universal Database, http://www-306.ibm.com/software/data/spatial/

2 Both Oracle Spatial and Oracle Locator are extensions to the Oracle Database 10g, http://www.oracle.com/technology/products/spatial/index.html

Representation of Locations When specifying the location information of entities in the virtual model different approaches can be taken into consideration. Depending on the requirement whether exact geographical information has to be preserved after the mapping is applied or not, separate common kinds of representing location data can be distinguished.

- *Symbolic coordinates* are used in the case where explicit position information is not available or not required. For example, an application which should switch on the light if at least one person is in the room does not need the accurate location coordinates of its users at all. It increments its number of persons in the room, whenever a user enters it, and decreases the number again if a user exits the room. If this number exceeds zero then it switches the light on, and turns it off if it equals zero.

  Symbolic coordinates should not be mixed up with conventional coordinate systems that define a fixed scale. Symbolic coordinates represent suitable unique identifiers like room and house numbers, postal codes or telephone numbers. A symbolic naming relation assign each spatial object, which should be included in the model, such an identifier. When applying the symbolic mapping relation to the three-dimensional features of physical objects their geographical coordinates are not preserved. Also symbolic coordinates do not support any spatial relations by themselves. If these relations are defined explicitly in addition to the symbolic mapping relation then a topological representation of spatial information can be obtained.

- *Topological representation* is the most frequently used approach when building context-aware applications. It consists of a symbolic naming relation, that is already mentioned, and a topological inclusion relation. The second one defines topological inclusion properties between the modeled items. Figure 12 depicts the persons Sara, Christine and Stephen having a meeting in room 11 on the 1st floor in the town hall.

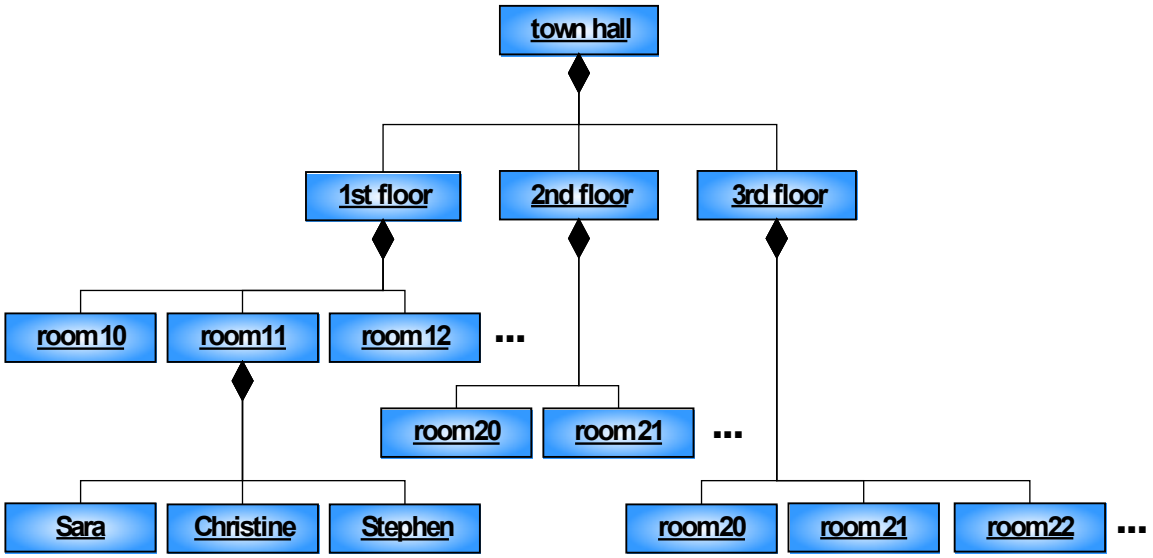


Figure 12: Topological relationships modeled with the UML composition pattern

If the context-aware application only needs to be informed about mere inclusion relationships, then the specification of position coordinates can be neglected. Already the ex-

ample above shows that topological representations are not very suitable for modeling an outdoor scenario. The higher the hierarchy level of an entity, the more inaccurate becomes its location information.

- *Geographic coordinate systems* are often used outdoors when position data is provided by GPS for example. In addition to the hight over sea level the position of the GPS receiver is specified in degrees of *longitude* and *latitude*. The *longitude* refers to the angle distance in degrees between the measurement point on the earth's surface and the Prime Meridian on the corresponding latitude. Since 1911 the Prime Meridian is defined as the first longitude line and goes through Greenwich, London. Locations in the east respectively in the west of the Prime Meridian are assigned an eastern respectively a western longitude between zero and 180 degrees.

  In contrast to it the *latitude* defines the angle distance between the referred measurement point and the equator on the corresponding longitude. Objects located in the north respectively in the south of the equator have a northern respectively southern latitude between zero and 90 degrees. The location of the GPS receiver is usually given in WGS84[1] coordinates. An extending standard is defined by ETRS89[2] which allows high-quality determinations of positions throughout Europe.
- *Projected coordinate systems* use projection rules to map points in a geographic coordinate system to Cartesian coordinates. Efficient algorithms are available for Cartesian coordinate systems that compute spatial relationships with respect to analytical geometry and the relations of position in three-dimensional space [MV01]. A common problem which is caused by the projection procedure is described in chapter 3.1 of [SHGN04]: "*Parameters of projected CSs[3] are determined to get the best attainable precision for a specific section of the earth's surface. Outside of this specific section the precision decreases constantly (and distortions increase). Thus a projected CS is characterized by a core and a peripheral area of use, with little projection errors in the core area, tolerable errors in the peripheral area, and significant errors outside the peripheral area.*"
- *Local coordinate systems* are often used by indoor systems and represent a special case of projected or Cartesian coordinate systems where the origin is mapped to a point within the building.

When modeling spatial information by integrating a coordinate system, the developer can choose from a large amount of standardized reference systems. The European Petroleum Survey Group (EPSG) maintains a common database of geodetic parameters that define 2824 coordinate reference systems and declarations of according transformations between these systems[4].

1 WGS84 stands for "World Geodetic System 1984".
2 ETRS89 terms "European Terrestrial Reference System 1989", visit "http://www.gps.gov.uk/gpssurveying.asp" for more information.
3 The abbreviation "CS" in the quotation denotes "coordinate system".
4 Further information about this topic and a MS Access database containing the referred parameters can be found on the Web Site of the EPSG at "http://www.epsg.org/".

Spatial Relations Besides location information the spatial context of entities is described by their spatial relations. This includes the distance to other persons or objects, their range, neighborhood relations and the inclusion properties that are already explained when talking about topological location representations. Considering spatial relations between different categories of moving entities there are statements which are of particular interest. For example, in the domain of automobiles various specific questions are important.

- Is the red car in front of the yellow one?
- If yes, does it represent an obstacle to the yellow car?
- If yes, does it represent a threat to the driver of the yellow car?
- If yes, how much time is left for a corresponding reaction of the yellow car's driver to avert it?

According to [RBB03] a context model can be classified by means of the spatial scope, the complexity of abstractions and the dynamics of the model. These three categories are explained as follows.

Spatial scope This category defines the physical range which can be represented by the model. That means the context model does only contain information of physical entities which are located in this area. The *spatial scope* can range from local small limited areas such as a smart desk in an office, a room or building up to large regions like an university's campus, a city, a country or as far as to the global scope.

Complexity of abstractions The *complexity of abstraction* refers to the detail of spatial context information contained in the model. A building that is modeled three-dimensionally with every corner and edge of each room represents a high complexity in contrast to a 2D bird eye's view of the mere conceptual structure of a particular floor in a building.

Dynamics Depending on the spatial scope and the model's complexity the term *dynamics* corresponds to the frequency of contextual information updates in the model of reality. Aware systems in home or office environments usually do not require a highly dynamic underlaying model since contextual changes do not appear as frequently as for example in an application domain where the safety of users is addressed.

Figure 13 shows this classification of context models along the three addressed dimensions according to [BN04].

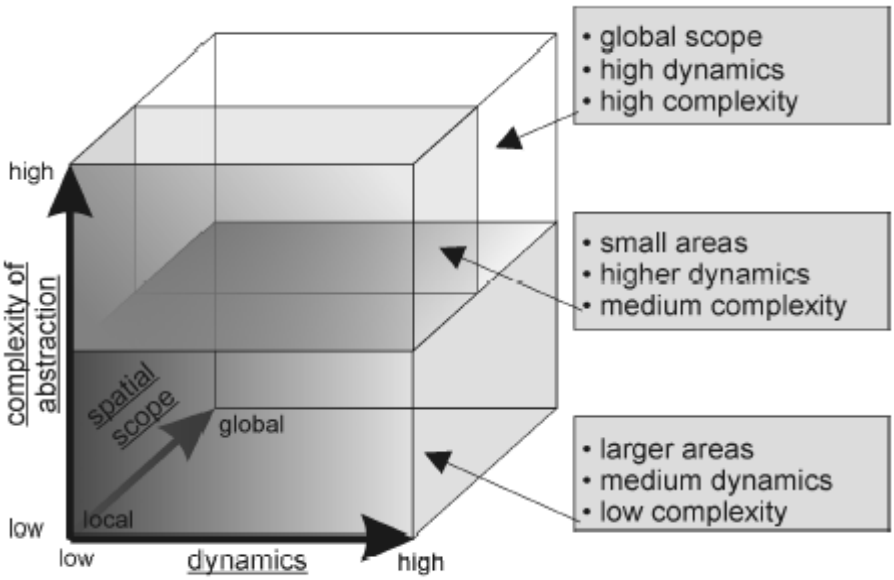


Figure 13: The three common dimensions of context models (picture from [BN04])

Representing spatial information using the addressed context models implies several benefits, but also some limitations which have to be concerned before deciding in favor of this approach when building context-aware systems.

Advantages Spatial context models are highly scalable with respect to the amount of information. In general there is no great difference in handling spatial information of only one entity or when managing hundreds of entities. For example, in the domain of driving assistance systems there is a need for modeling a high amount of vehicles, road information and objects such as traffic signs or location information of interesting spots. Here not only the local environment has to be modeled but also large areas in order to reason about traffic jams for instance. High performance is also a feature of spatial context models. So in general they can cope with a high frequency of updates. Another advantage is the efficiency in spatial queries as these systems generally incorporate an explicit model of space. Range queries and nearest neighbor searches can be done with high performance due to the underlaying coordinate system.

Disadvantages A common problem of spatial context models is the restriction of knowledge sharing with other management systems because transformation rules of spatial data between different representations have to be implemented explicitly. By integrating standardized interfaces this drawback can be resolved. When modeling spatial context, apart from the mere representation of the entities' locations, common and also specific spatial relations have to be concerned that are subject to versatile context-aware application domains. It is hard to abstract spatial context models this way to provide a common basis that satisfies all possible interpretation requirements of applications. Therefore specific statements about do-

main knowledge have to be investigated by the applications themselves. However the sharing and reusing of reasoning results is often desirable.

A different approach in context management is represented by contextual ontologies which enable applications to additionally understand the meaning of contextual information.

2.4.2. Contextual Ontologies

Conventional approaches are aimed at merely providing structured data sets. In contrast *contextual ontologies* additionally support semantic information about the context data. Therefore they represent a powerful instrument for the management of contextual information.

To get familiar with the term "ontology" let us fist look at it from a philosophical viewpoint. "*Philosophical ontology is the science of **what is**, of the kinds and structures of objects, properties, events, processes and relations in every area of reality.*" [SW01]. According to [FEL98] an ontology represents "[...] *the metaphysical study of the nature of being and existence*".

With respect to the area of computer science Gruber states that an "[...] *ontology is a description (like a formal specification of a program) of the concepts and relationships that **can exist** for an agent or a community of agents."* [GRU05]. When comparing both definitions I would like to call attention to the marked words. The difference between "what is" and "can exist" denotes the different points of view of both researchers. The philosopher Smith states, that there is exactly one ontology describing everything of reality. In contrast Gruber bases his definition on parts of reality which can be observed. Bases on an observed reality he restricts the term "ontology" to the knowledge which "can" be attributed to agents. It implies that there is more than one ontology. This is due to the fact that an observation of reality is always restricted to one of its endless subsets.

There is no better way of describing reality than on the basis of observation. This fact will be further investigated when the tiers of ontology are presented later in this chapter. Smith's definition does not address the area of computer science. However Gruber's definition is also not applicable since by involving concepts and relationships of agents this does not necessarily involve their individual properties.

Another explanation of the term is given by the W3C[1] which explains in [W3C04a] that an ontology "[...] *defines the terms used to describe and represent an area of knowledge. Ontologies are used by people, databases, and applications that need to share domain information* [...]. *Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them* [...]. *They encode knowledge in a domain and also knowledge that spans domains. In this way, they make that knowledge reusable."*

By means of this description I would like to propose an appropriate definition as follows.

1 **W**orld **W**ide **W**eb **C**onsortium

Definition 4: Ontology in Computer Science

*An **ontology** is a conglomeration of structured and organized concepts and assertions, that describe knowledge in selected observable domains of interest. Concepts are represented by their mutual dependencies and relationships as well as by their particular properties.*

Ontologies can be used to describe knowledge of an application domain in terms of classes, relationships between entities, and entities (instances of a class) themselves together with their individual properties. It is possible to distinguish between *terminological knowledge* regarding the common concepts and *assertional knowledge* referring to the information about individuals in the world.

The structured ordering of ontological classes allows for multiple inheritance, which means that an entity can inherit a (possibly restricted) set of properties of more than one super class. Inheritance can also be applied to relations between classes. This extends the hierarchical organization and enables a powerful mechanism to synthesize knowledge.

As an example for an ontology imagine one called “mobile things” as shown in figure 14. Similar to the declaration of object classes in UML, ontology classes can be assigned relations and attributes. Relations between classes are always of binary type. Properties of a class represent their attributes, and class entities are shown as instances.

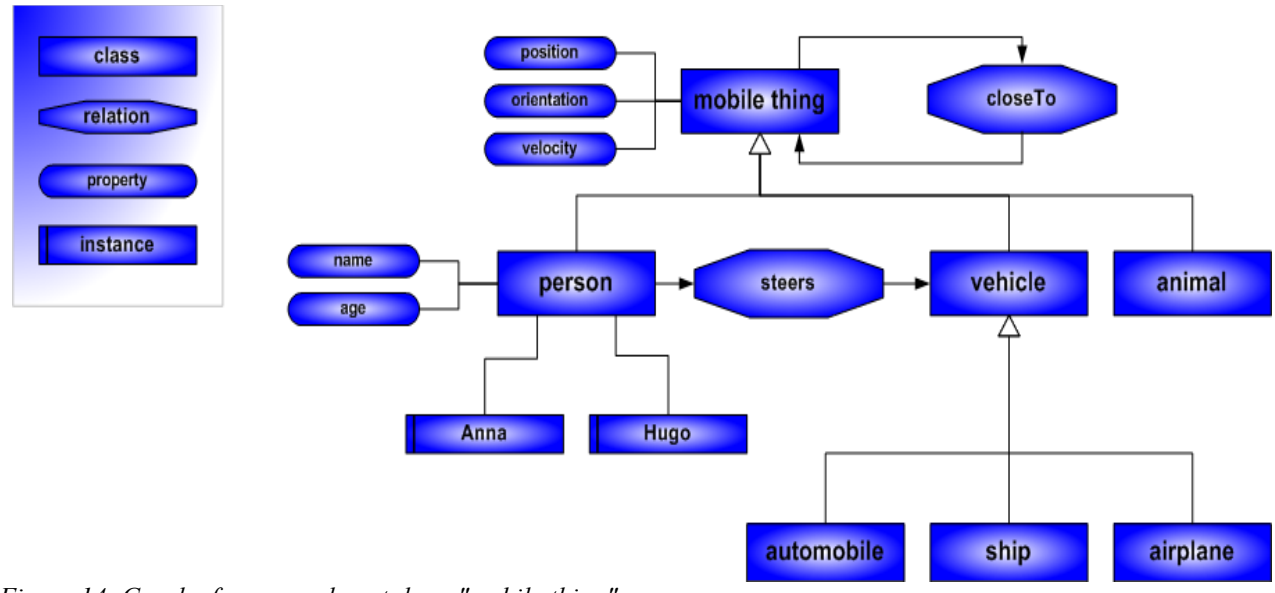


Figure 14: Graph of an example ontology "mobile thing"

As declared in the ontology the persons Anna and Hugo are “mobile things”, and have the properties “name”, “age”, “position”, “orientation” and “velocity”. The binary relation “steers” tells that a person can steer a vehicle. Finally another relation “closeTo” can be applied to all classes mentioned in the ontology because it is declared for the “mobile thing” class that is a super class of all others.

In general the specification of ontologies follows standardized modeling languages. Therefore an ontology-based knowledge representation can be composed of separate resources which may be created by different developers and shared amongst separate context-aware software components. This allows a common understanding of contextual information and its semantics independently of the hardware platform the components are deployed at or programming languages they are implemented with.

Ontologies can also be combined in order to extend the knowledge space. With respect to computer science I call the resulting structure an ontology space.

Definition 5: Ontology Space in Computer Science

*An **ontology space** is represented by a composition of areas of domain knowledge. Each dimension of the space is defined by an item of the intersection set containing the distinct ontologies within the space.*

When extending the “mobile thing” ontology by an ontology that describes the relations of a road network in a country an ontology space is created of the two dimensions mobility and navigation. For example, this newly created ontology space enables applications to provide routing information to drivers.

Before talking about knowledge deduction with respect to ontologies, let us first have a look at the five tiers of ontology which are defined by Frank in [FRA03].

Ontology Tier 0: Physical Reality *reality :: world -> property -> spacePoint -> timePoint -> value*
Ontology Tier 1: Observable Reality *observation :: world -> observationType -> location -> measurement value*
Ontology Tier 2: Object World *observation :: id -> time -> observationType -> value*
Ontology Tier 3: Social Reality *getName :: object -> name* *findObject :: name -> environment -> object*
Ontology Tier 4: Cognitive Agents *rules used or deduction*

Table 2: Five tiers of ontology

Tier 0 describes physical reality in general. It is based on the assumption that exactly one physical world exists, and that it contains time and space as its dimensions. The function “reality” provides for every property in the one and only world and for a fixed point in time and

space a single value. This value is represented by real numbers and can either be quantitative or qualitative.

Tier 1 defines the limited subset of reality which is observable with limited methods at a present moment. Observing the world over a period of time results in a set of observations at discrete time points. A single measurement value is retrieved with a certain observation type at a particular location. The type of observation implies the scale and unit of the result of the observation process. This measurement value is never accurate since observation errors can not be avoided. Note that the physical world is solely observable. This is due to the fact that everybody who tries to describe reality represents implicitly also just an observer.

Tier 2 encapsulates distinct observations into a description about individual objects. These objects are restricted to physical ones by Frank. They are characterized by their dynamic boundaries and areas of uniform properties which are related to the type of the interaction with them. For example, uniform properties represent the same color, type of material or type of movement. In contrast to “volatile” observations of the real world objects preserve their state of properties over a period of time. This allows to use individual identifiers to address objects. An object is represented by a function for which an identifier, a type of observation and a time value has to be specified.

Tier 3 creates a domain of social relationships as they are perceived by human beings. The essential fact of social interaction is the ability to reference the objects of tier 2 by names. Although physical reality does not care about naming things these social aspects appear as real to us. This tier defines functions to retrieve the name of an object and to find an object by its name in a specified context in which the name is valid. For example, this context can be institutional, administrative or legal.

Tier 4 describes the modeling of cognitive agents. A cognitive agent is an entity which gathers and aggregates context about both the physical and the social world and performs actions depending on its declarative knowledge. Besides directly observing reality, an agent also retrieves knowledge from other agents. In order to make decisions an agent can lookup the acquired information or infer new knowledge by rule-based logical deduction.

Except for tier 0 all other tiers can be accessed by context-aware applications. With the help of deduction rules context management systems can be developed that implement tier four where agents – and thus also software systems – are used to deduce contextual information.

Context management systems using contextual ontologies are able to identify inconsistencies in their knowledge representation. In addition, the reasoning subsystem of a context management architecture can implicitly deduce new from already existing knowledge. This enables the reasoning about common relationships of entities. Context-aware applications require to process this implicit contextual data. However the information, which is acquired from sensor systems, is only available in an explicit type. Using logic based approaches implicit context information can be deduced from explicit low-level data. The information which is contained in ontologies represents declarative knowledge. Therefore description logic is well suited and generally used for the representation of contextual knowledge.

2.4.2.1. Description Logics used for Ontology-based Context Reasoning

Description logics cover a family of languages that are employed to declare knowledge about concepts and their hierarchies with respect to an application domain. Their basic structure is set up by concept classes, binary relations between individuals of classes and individuals. Concept classes encapsulate the common properties of a set of individuals and specify general relationships between them. Language constructs are used to define the structure of a particular description logic furthermore. These are intersection, union, transitivity, complement, enumeration, disjunction, equality, inequality, etc.

Description logics can be used to evaluate the satisfiability and consistency of the modeled concept, class subsumption and the validation of instances, i.e. individuals of a concept class. For a comprehensive insight into the theory of description logics the reader is referred to [BCM03], [MB95], [BOR95] and especially [CLN98].

When comparing description logics to the constructs of ontologies it can be seen, that both structures are quite similar to each other. Concept classes can be mapped to ontology classes, binary relations to ontological properties and individuals to ontological entities. This enables logic-based reasoning over ontologies.

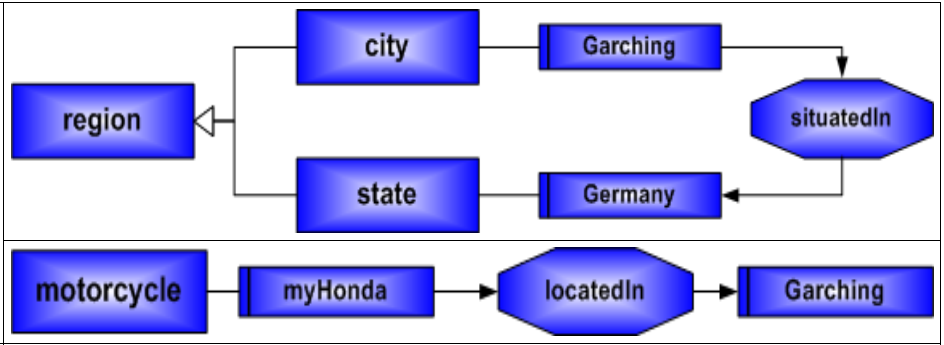
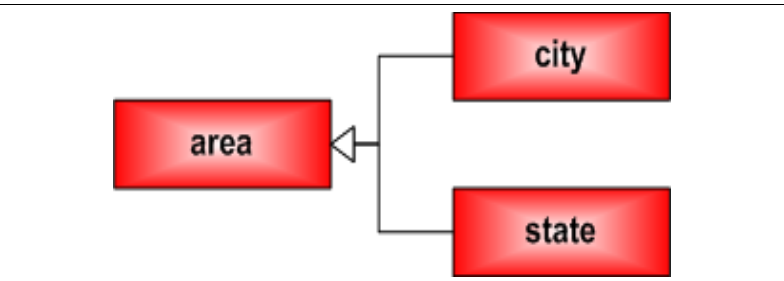
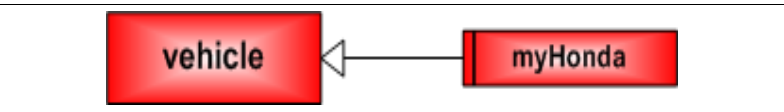
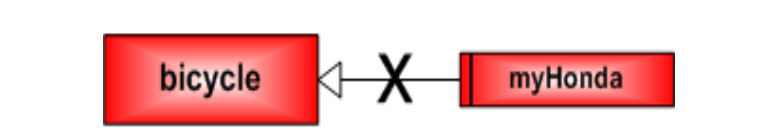
Consider the example properties in table 3:

symmetry	let S be a symmetric property if (a, b) is an instance of S ⇒ (b, a) is an instance of S
transitivity	let T be a transitive property if (a, b) and (b, c) are instances of T ⇒ (a, c) is an instance of T
inversion	let I1, I2 be properties and I1 is inverse of I2 if (a, b) is an instance of I1 ⇒ (b, a) is an instance of I2
synonym	let d1, d2 be properties and d1 is synonymous to d2 if a is an instance of d1 and b is an instance of d2 ⇒a is same as b
disjointness	let d1, d2 be properties and d1 is disjoint with d2 if a is an instance of d1 and b is an instance of d2 ⇒a is different from b

Table 3: Exemplary reasoning rules

Applied to the ontology in figure 15 one can see how these language constructs are used to model and deduce knowledge for a particular domain. Let us declare some spatial information in table 4 which is based on this figure, and see what implicit knowledge an inference engine could deduce from it.

Table 4: Description logic-based ontology reasoning

<table>
<tr><td>explicit information</td><td> 
</td></tr>
<tr><td rowspan="5">implicit information</td><td> 
</td></tr>
<tr><td> 
</td></tr>
<tr><td> 
</td></tr>
<tr><td> 
</td></tr>
<tr><td> 
</td></tr>
<tr><td> 
</td></tr>
<tr><td> 
</td></tr>
</table>

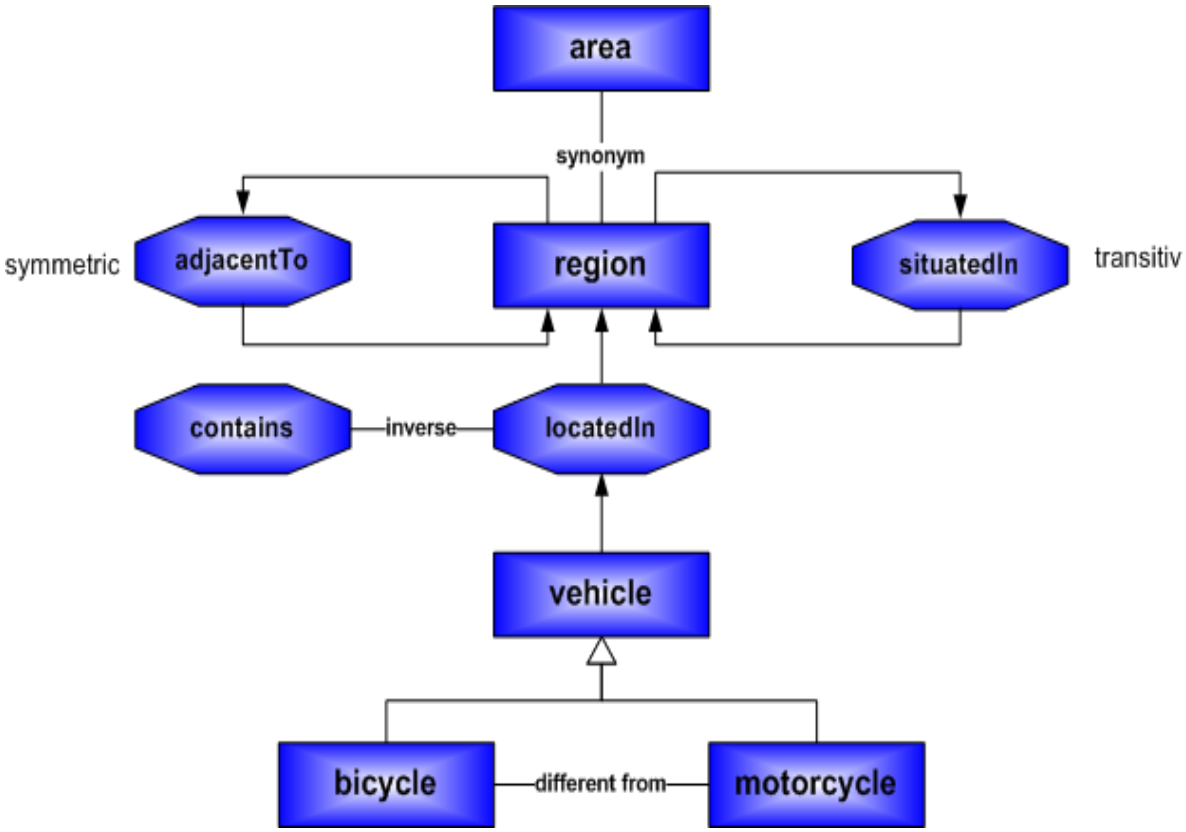


Figure 15: Example ontology demonstrating language constructs

When looking at figure 15 the usage of the example language constructs of table 3 can be seen. These have been used in the reasoning process depicted in table 4. The symmetric property “adjacentTo” has not been explained there. It states that, for example, the region “Germany” is adjacent to the region “Austria” if the property holds the other way round, too.

The transitive property could also be applied to the subclass relationship. With the knowledge that “bicycle” is a subclass of “vehicle” and “vehicle” is a subclass of “mobile thing” the logic based reasoner can deduce that the class “bicycle” also has the subclass property with respect to the “mobile thing” class.

These examples demonstrate how an inference engine is capable of deducing additional knowledge by applying logic-based reasoning to ontologies. The combination of basic ontological information and the reasoning results form a so-called *ontological knowledge base*.

Definition 6: Ontological Knowledge Base (OKB)

*An **ontological knowledge base** is a volatile or persistent data repository containing a collection of ontological statements and possibly the information that is inferred in the corresponding ontology space.*

On the one hand the OKB can be used locally in the scope of individual applications. On the other hand it can also be extended and shared among different systems. Therefore reasoning results of contextual information is made accessible for other applications.

2.4.2.2. User-defined Context Reasoning

Description logics can be used to model the contextual information and the common relationships between classes of individuals. This declarative knowledge is often of a lower-level contextual type. However aware applications also require reasoning about the user's behavior and situational environment. They require additional knowledge about the application's domain.

A common approach is the combination of ontology-based reasoning with *user-defined reasoning*. By applying user-defined context reasoning to ontological information higher-level context can be acquired. Basically there are two different ways of accomplishing user-defined reasoning.

Machine Learning On the one hand different approaches of machine learning may be applied. Among other techniques neural networks, Bayesian and reinforcement learning can be listed here. Neural networks are derived from the pattern of the synaptic structure of the human brain, and are employed when reactions to different contextual conditions have to be learned interactively. Like neural networks also reinforcement learning allows the acquisition of knowledge for proper actions according to frequently appearing contextual configurations. Bayesian learning can be used if information about the probabilities of occurring context events has to be analyzed.

Machine learning is often not very suitable for deducing highly dynamic spatial context due to the complex learning processes. Above all in situations where an implementation does not have the required “experience” at its disposal to decide precisely about the contextual information it most likely will interpret inaccurate or ambiguous and therefore frequently not usable data, that could have been efficiently inferred using a set of simple rules. However in contrast to spatial information machine learning techniques are more qualified when reasoning about high-level context like the user's social situation. For example, it is extremely difficult and awkward to specify rules to deduce information about the user's mood which is dependent on various situational circumstances and preconditions, and also changes noticeable from person to person.

Rule-based Approach On the other hand rules are used which can be formalized in various forms of logics such as description logic, first order logic, higher order logic, temporal logic, fuzzy logic and so forth. As explained before description logic allows reasoning about hierarchies of contextual knowledge. First order logic is often used when dealing with universal and existential quantifiers to issue generic statements. If more expressive power is needed higher

order logic may be appropriate. Temporal logics like the branching-time logic CTL[1] or the Linear Temporal Logic (LTL) are used to evaluate the temporal occurrence of contextual events. If the representation of context information includes a lot of uncertainties then some kind of fuzzy logic is a good choice. The rule-based approach is appropriate if the reasoning process is based on terminological or frequently occurring contextual information which describes a generically understandable domain of knowledge. Therefore it is well-suited to deduce spatial context information which are perceived and recognized by different (human) observers in the same way and obey common rules.

To give an example of user-defined context reasoning let us focus on first-order predicates to define deduction rules for the process of inference. Such a first-order predicate is defined by a triple which is composed of a subject, a predicate and an object. For example the informal statement “Anna is in the car” can be represented by the triple (Anna, locatedIn, car). The following table shows some examples of a spatial reasoning process that is dependent on the according condition describing the user's situation.

user's action	condition	reasoning result
slowing down car	*(user, steering, car)* *AND* *(car, situation, DRIVING)* *AND* *(user, pressing, brakePedal)*	$\Rightarrow$ *(user, decelerating, car)*
overtaking with car in right-hand traffic	*(user, steering, car)* *AND* *(car, locatedBehind, vehicle)* *AND* *(user, setting, leftTrafficator)* *AND* *(user, switching, lane)*	$\Rightarrow$ *(user, overtaking, vehicle)*

1 **C**omputation **T**ree **L**ogic

user's action	condition	reasoning result
crashing into another vehicle	*(user, steering, car)* AND *(car, locatedBehind, vehicle)* AND *(vehicle, isObstacleFor, car)* AND *(car, timeToCollision, FATAL)*	⇒ *(user, crashingInto, vehicle)*

Table 5: User-defined context reasoning about spatial information using first-order predicates

As one can imagine, the reasoning rules in table 5 involve a high degree of contextual uncertainties. The condition for the second example (overtaking) also holds for a driver who does not want to overtake another vehicle at all. When the driver wants to turn left at the next road intersection, and switches to the left lane to slow down at the traffic lights then the system would misinterpret the contextual input. In order to reduce ambiguity when performing reasoning additional triples have to be added to further limiting the condition in order to address a specific situation of the application domain.

As follows usability aspects of contextual ontologies are discussed with respect to spatial context information.

Advantages The key feature of contextual ontologies is represented by their reasoning capabilities. In contrast to falling back on limited implemented deduction algorithms they enable inference by means of ontologies which can be easily accessed by different systems. In addition to identifying inconsistencies contained in ontologies the reasoner can deduce new contextual information from existing knowledge. The gained knowledge can also be shared amongst different context-aware applications at run-time, and general concepts can be easily reused when developing new applications, thus enabling rapid prototyping.

Disadvantages Context management systems using ontologies are not suitable for large amounts of data such as provided on the ontology tiers one to three. Especially high update rates of spatial information which are easily handled by context models result in complicated explicit information representations within the ontology space. As stated in [MH97], "*Ontologies can grow and shrink as necessary based on the context where they are being used.*" However, this statement only applies to environments where the ontology space is changed quite frequently or where the amount of context data does not accumulate too much. In systems where highly scaled contextual information has to be processed sooner or later this increases the OKB to an extent which can not be be processed efficiently by applications any more. Compared to context models context management systems merely employing ontologies are not distinguishing themselves when dealing with high access frequencies that are re-

quired by context-aware applications addressing rapidly changing environments.

By combining both methods of context modeling their disadvantages can be reduced to a neglectable degree. The following chapter introduces the reader to such an approach.

2.4.3. A Combined Approach

In [BN04] Becker and Nicklas propose a context management architecture which "[...] c*ombines the strengths of both approaches while trying not to carry the specific weaknesses into the resulting architecture* [...]". Figure 16 depicts the arrangement of the context management subsystems along the ontology tiers one up to four which can be accessed by computer systems.

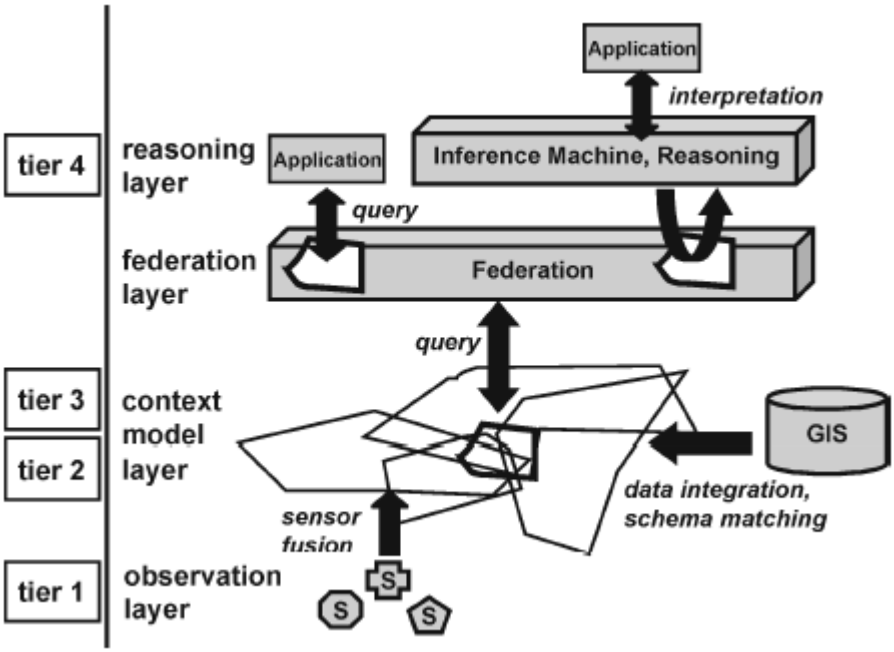


Figure 16: A combined approach according to [BN04]

On tier one sensors observe the physical reality and provide basic contextual information to local context models. Tiers two and three act as a mapping layer which combines the different information representations of the local models and integrates data coming from geographical information systems. The federation subsystem above acts as a facade pattern and controls access to the integrated context models. A querying mechanism enables context-aware applications to directly access low-level contextual data via the federation layer or retrieve higher-level implicit information from the reasoning subsystem on tier four. The latter subsystem also implements the facade pattern thus handling access control and making the reuse of deduced contextual information possible.

The idea behind this approach is based on the fact that context management architectures

are aimed at supporting various aware applications with different requirements according to the spatial scope, the complexity of abstractions and the dynamics of the modeled context.

Ontologies can be used to deposit and deduce information about the common concepts of different application domains. The reasoning subsystem additionally receives up-to-date information from one or more spatial context models which are seamlessly integrated via the federation subsystem. These models are well-suited to incorporate high amounts of low-level context data such as basic spatial information together with high update rates (dynamics). The context management architecture has to process explicit knowledge with a fine degree of granularity, and provide inferred and abstracted context information to aware applications, whose decision subsystems are optimized with respect to limited spatial areas.

For example, applications do not have to query the reasoning subsystem about the user's accurate position and orientation if they just want to know whether or not a person is located in a room. However the management systems are dependent on an efficient knowledge representation such as spatial models based on coordinate systems. Therefore the most important task is the mediation between explicitly modeled spatial information and their abstraction in terms of ontologies.

Connecting Ontology-based Reasoning and Spatial Context Models In order to determine the interface between the two combined approaches I propose to base the user-defined reasoning of ontological information on one or more underlaying spatial context models.

The key essence of this approach is to base the decision of whether or not a terminologically declared relation between particular entities holds in a certain situation on the data, which is contained in highly dynamic spatial context models.

This implies the idea to separate the concept knowledge from the assertional knowledge, that is acquired from the individuals in the real and also the virtual world, or may also be available as historical information. Each binary relation between ontology classes can be represented with the help of software functions which access the spatial context models either directly or via the mediation capabilities of a federation layer. These functions are referenced in the specification of user-defined rule sets to compute the dynamic spatial relationships between entities. In this case higher-level context are deduced according to the terminologically declared binary relation that refers to the entities' classes.

For example, in figure 17 on the next page the relation “isSpatialObstacleFor” tells whether a “spatial thing” (every not movable entity that is registered by its location) represents a spatial obstacle for a “mobile thing” (every movable “spatial thing”). In order to respond to abstracted spatial queries regarding this relation, information of one or more spatial context models is interpreted with the help of implemented or dynamically compiled functions according to user-defined rules. The reusable functions can be employed to process the spatial data of the entities and the relationship between them. Such a functions could respond the information how close an entity is to another one or whether an object is in front of an entity. The combination of these functions enables the deduction of higher-level spatial context.

For instance, with respect to the “isSpatialObstacleFor” relation the reasoner is now capable

of deducing that a person, who abruptly jumps in front of a car represents an obstacle for the driver. For the purpose of reasoning about historical context, and the management of persistent information, the deduction results can be stored as assertional knowledge for later retrieval.

The advantage of the combined approach is the incorporation of general concept specifications by means of ontologies and the employment of efficient and detailed spatial context models containing up-to-date information about the ontological entities. Depending on the accuracy of the spatial knowledge representation by the context models, spatial data of static and moving objects can be interpreted precisely on the reasoning layer. If the addressed functions are specified in a generic way they can be reused when building applications. Adding new functions to the spatial context architecture extends the power of the function library, which itself can be seen as a knowledge base for user-defined reasoning rules.

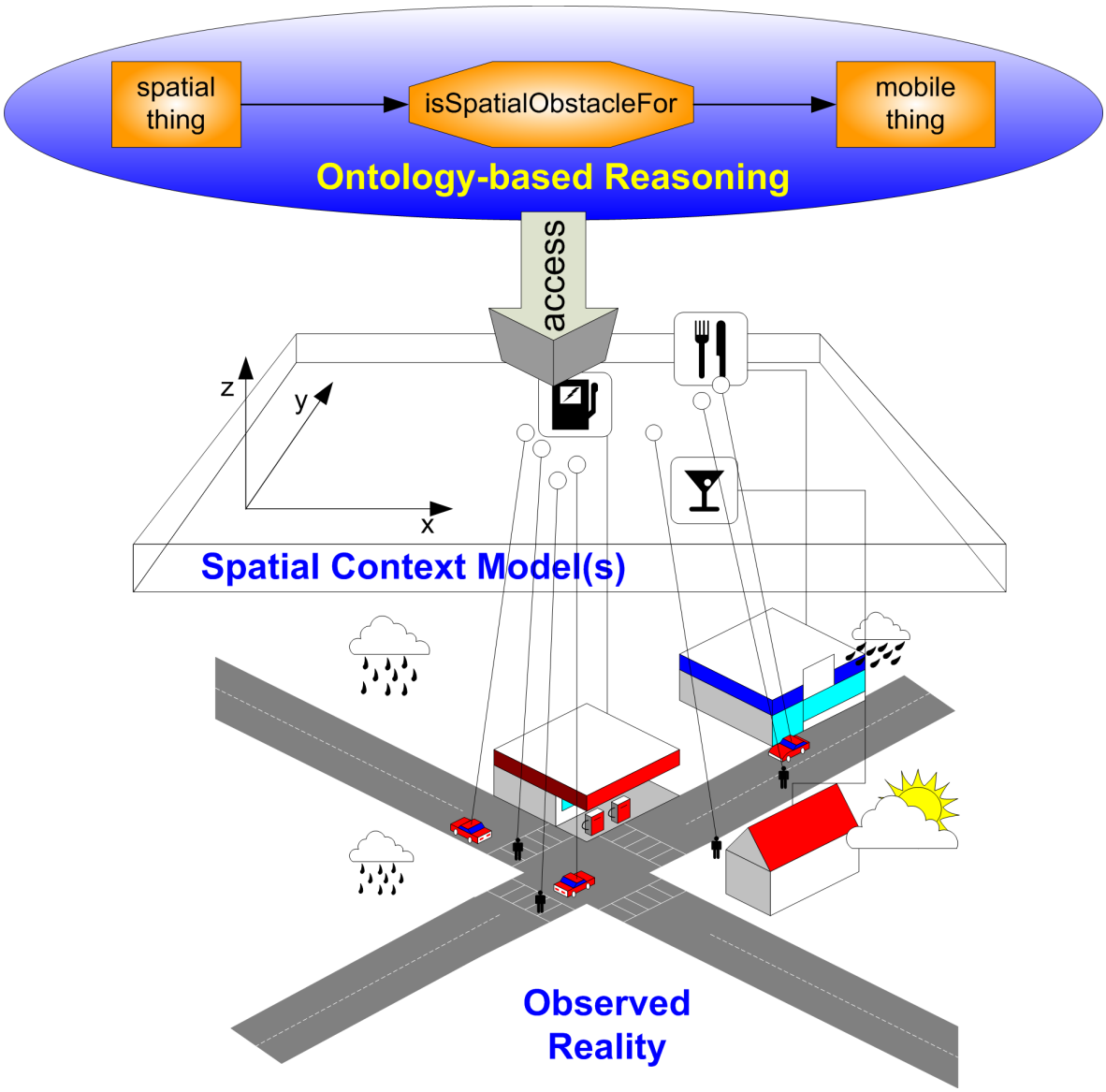


Figure 17: Ontological class relations dynamically defined by properties of spatial context models

3. Related Work in Context Management Frameworks

The previous chapter explained different approaches of modeling, representing and reasoning about context with special focus on location-based context information. Reusable software architectures that pre-process contextual data for different aware application are called *context management frameworks*. Before a deeper view on the fundamentals of selected context management frameworks is provided in order to understand the architecture, benefits and limitations of these systems, the general concepts of frameworks are briefly discussed.

According to the on-line encyclopedia “Lockergnome”[1] a software framework in computer science is defined as follows.

> **Definition 7: Framework**
>
> *A **framework** is a defined support structure in which another software project can be organized and developed.*

Advantages By interacting with a framework, applications can profit from its capabilities and its already tested stability and robustness. The use of prefabricated components decreases the time of development for new applications, which just have to be customized to the application domain they are designed for. Therefore incorporating frameworks decreases the overall costs for the implementation of applications, and with respect to commercial products it also shortens the time-to-market and enables parity with competing products.

Disadvantages On the other hand also limitations are involved which have to be considered by software engineers when selecting an appropriate framework for a specific software system. Frameworks are often complex software architectures. Investigating the framework's structure and components, the necessary programming techniques, design patterns and the abstracted control-flow requires a lot of learning time, which should not be underestimated. Furthermore frameworks are distinguishing themselves by the kinds of applications they support. Platform-independence is not necessarily a prerequisite for framework development, and often there are also limitations to the interaction and communication techniques between the application and the framework. Finally the framework itself requires computing resources in addition to the system, that uses it.

When recalling the requirements for context management in chapter 2.3, it is possible to say that the principal tasks of frameworks, that are dedicated to the management of contextual information, are represented by...

- the structured acquisition, aggregation and interpretation of context information,
- the disintegration of contextual inconsistencies,

1 http://encyclopedia.lockergnome.com/s/b/Framework

- the persistence management for storing and recovering context,
- the support for distributed and transparent communication, the discovery of context resources and
- the guarantee of safety and privacy.

Not all of these topics are addressed by existing context management frameworks. Above all, the last requirement, which describes safety and privacy concerns, is only addressed with respect to the security of sensitive and personal context data, because the user's safety within hers environment is mostly a topic of the applications. Therefore the context management frameworks, which are described in the succeeding sections, will be discussed with respect to the requirements mentioned above except for the safety concerns.

According to my intensive investigations currently there is no context management framework, which is really *specialized* in the handling of higher-level spatial context, and that is dedicated to process abstracted spatial queries. Therefore the following sections mainly discuss related systems in the area of general context management, which of course also includes explicit location-based information.

As follows the three different software frameworks Nexus, CoBrA and the Context Toolkit are investigated. They are aimed at facilitating the design and implementation of context-aware applications. The most interesting aspect of these frameworks refers to their different approaches with respect to their architectural styles and their representation of contextual information.

3.1. Nexus

The interdisciplinary research platform *Nexus*[1] is developed at the University of Stuttgart and represents a layered architecture for mobile context-aware and Augmented Reality applications. Though the Nexus platform is able to deal with context information in general, its principal focus relies on the management of *explicit* spatial context information.

Nexus is aimed at integrating location-based information which is acquired from different distributed data sources, and provides a standardized interface to applications requesting this information. Here the data is accumulated in different local spatial context models. To enable a uniform and transparent information access, the platform supports the federation of these models into a global spatial context model, the so-called Augmented World Model. This global context model of Nexus can be classified according to the three dimensions of location-based context models (see figure 13), which have been introduced in chapter 2.4.1.

Classification The spatial region which can be covered by the context representation corresponds to the *global scope*. Therefore both indoor and outdoor applications can be built on top of Nexus regardless of whether they are used in areas with limited extent or in highly

1 http://www.nexus.uni-stuttgart.de/index.html

scaled environments. The Augmented World Model also supports a *high complexity of abstractions* and *high dynamics*. That means that on the one hand objects can be represented with a very high degree of detail, and on the other hand a high frequency of model updates is supported.

3.1.1. Architecture

As depicted in figure 18 the architecture of the Nexus platform is divided into three layers. The top layer contains software applications which make use of Nexus. The middle layer is responsible for mediating between applications and context servers that contain the local context models. Finally the context servers are placed on the bottom layer, and may cover spatial information of both the real and virtual world in geometric and / or symbolic context representations [ZEI04].

In [VOL01] the author explains the idea behind the architecture of Nexus with the help of an intuitive relation: "*Compared to the World Wide Web (WWW), the applications within the top layer can be seen as web browsers and the servers within the bottom layer as HTTP[1] or FTP[2] servers. The middle layer has no equivalence in the WWW architecture.*"

The aim of the layered architecture is the decoupling of context providers and consumers thus allowing the integration of new context data sources without having to modify or adapt the implementation of yet existing applications on the top layer.

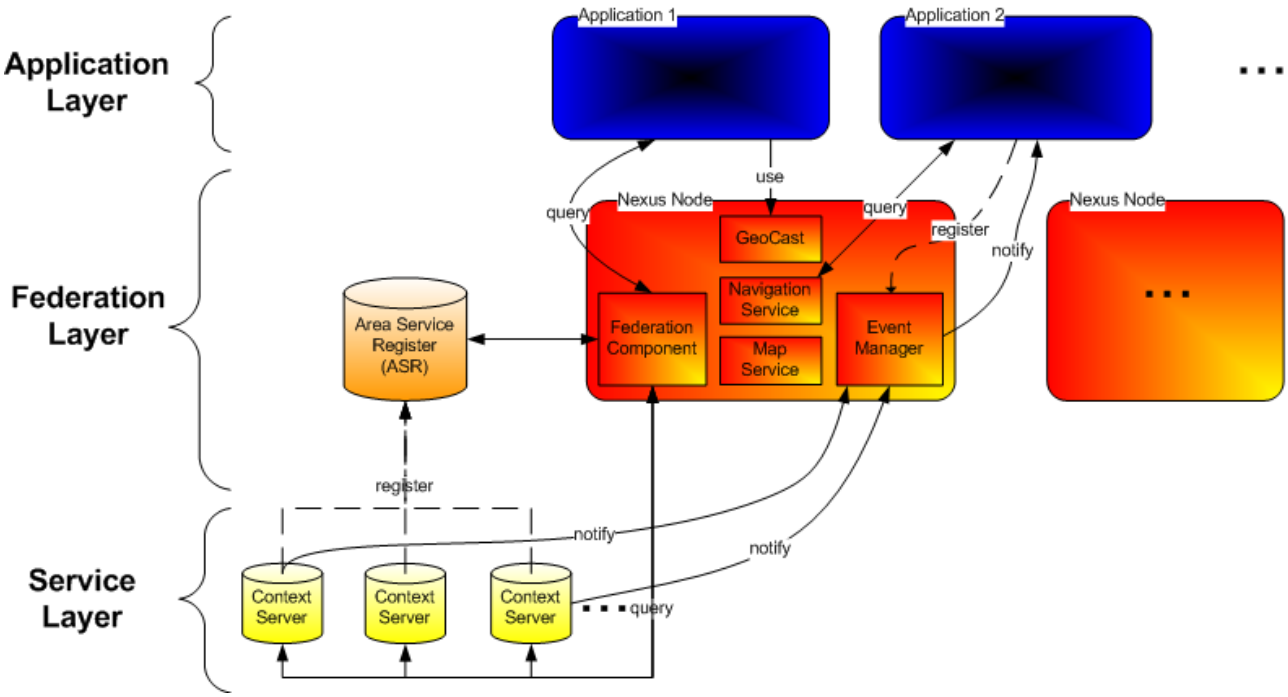


Figure 18: Layered architecture of Nexus

1 **H**yper**T**ext **T**ransfer **P**rotocol
2 **F**ile **T**ransfer **P**rotocol

Application Layer On the *application layer* the Nexus platform enables applications independent of a certain platform, which run on mobile or stationary devices and can be situated in the fields of context-awareness and Augmented Reality. They can either use the mediating components on the federation layer or directly access services on the bottom layer by means of standardized programming interfaces.

Using the federation layer applications are able to issue queries to retrieve spatial information about modeled entities within their environments. It is also possible to register applications with an event mechanism so that they are notified on certain events whenever selected contextual attributes have changed.

For example, applications may subscribe to events that are triggered if certain entities like persons or cars are leaving a particular spatial area. In addition to the query and event notification mechanisms on the federation layer, applications can also contact so-called value-added services.

Federation Layer These value-added services encapsulate common application tasks like navigation and routing or the customizable creation of geographical image or vector maps from spatial context models. Together with the event manager and a federation component, that coordinates and caches application queries regarding the local context models on the bottom layer, value-added services are encapsulated in a Nexus node residing on the *federation layer*.

Nexus nodes can be distributed over the existing computational infrastructure and are selected by applications usually according to their proximity to the user's position. The Nexus node, or rather the federation component has to know which of the context servers containing the local context models should be accessed on a particular query.

This knowledge is obtained from the *Area Service Register* (ASR) that holds meta-data information about all local context models, service discovery data of their managing servers, the types and unique identifications of contained entities and the spatial area which is managed by each model. The ASR represents the “yellow pages” for searching spatial information in distributed context servers.

Another value-added service, the *GeoCast* system can be used by applications to send multicast messages to registered devices in a given spatial area. For example, at midday all users who are located close to a university's cafeteria can be informed about the lunch menu this way.

Nexus also supports an information hoarding component that pre-fetches context data based on a statistical evaluation of what information might be of particular interest to the user at her current location [KR01]. The context items are transfered to the user's mobile computer in advance before a user actually demands access to them. Finally, if the information is required, it does not need to be queried, but can be displayed immediately.

The publish/subscribe mechanism of the event managing component allows the abstraction of messages by decoupling senders and receivers. The event manager is notified about contex-

tual changes in the local context models on the service layer and informs all applications which have subscribed to according events.

Service Layer As mentioned before location-based information about real and virtual objects is stored in local context models. These models are managed by context servers located on the *service layer*, and can be updated by connected sensor systems. To enable the Nexus nodes to query context servers or be notified about context changes via events, each context server has to register with the Area Service Register and announce its meta-data information to it.

In general there are two different types of context servers in the Nexus platform. Above all *Spatial Model Servers* contain static location-based information of often large geographical regions. They incorporate database management systems which are extended and optimized for spatial queries. The other principal type of a context server is the *Location Server*. Using distributed main memory it is better suited to deal with high dynamics of context models which contain moving entities.

In [BDG04] the authors state that also other types of context servers can be integrated into the service layer if they meet the requirements for the implementation of the standardized interface to the Nexus platform and the registration with the ASR.

3.1.2. Discussion

The Nexus platform manages the integration of distributed context information, which can be accessed by applications in a uniform way. This is achieved by the three-layer-architecture with the federation layer located in the middle, and by employing standardized interfaces.

The layered architecture enables scalability, distribution, modularization and – with respect to applications – information hiding. Information hiding means that data which is irrelevant to an application on the top layer is not published to it. Logical data independence allows modifications to the service layer without any consequences for the applications on the top layer.

The architecture supports synchronous communication via query interfaces on the bottom and middle layers. Its disadvantage is the blocking behavior in the control flow of the querying service while the query is processed. Also changes to the communication interface directly affect the querying component and the service which implements the interface.

Besides synchronous communication the architecture allows for asynchronous event notifications with the help of event managers and the GeoCast service on the federation layer. In [DBF04] the authors believe that “[...] *the publish/subscribe or event based communication paradigm is well suited to address the issues of pervasive systems in a mobile ad hoc environment.* [...] *In addition, asynchronous communication is ideal in systems where frequent disconnections are likely to happen thus to avoid blocking conditions.”*

Event-based communication also implies considerable limitations. Distributed event handlers are making the synchronization and both debugging and validation of components more diffi-

cult. Obsolete or invalidated events have to be detected and treated properly. Without additional explicit synchronization, it is not possible to process parallel events in a particular order.

Context Acquisition and Aggregation Each context server on the service layer is responsible for integrating location-based information into a local context model. The contextual information is aggregated in these models and federated into a global model. This global model does not contain copies of context data but is able to reference the context servers as data sources for particular information. A frequently occurring situation is the handover of the responsibility for mobile entities between local context models whenever an entity crosses the boundary of a region that is covered by such a model. This handover mechanism is yet intended to be implemented.

A terminological representation to declare common concepts and relationships between aggregated contextual information of entities is currently not supported.

Context Interpretation and Disintegration of Contextual Inconsistencies Both the reasoning and handling of contextual inconsistencies and ambiguities is shifted towards the application layer. Neither reasoning results are shareable and reusable by other applications nor they are saved as past contextual information for later retrieval.

Persistence Management The separation between static and dynamic spatial context data on the bottom layer helps mastering the flood of information. Static context information is stored in exchangeable spatial database management systems which provide an efficient query mechanism for location-based information. However data about moving entities is held in main memory and is therefore volatile. Also all context information processed on the federation layer is not persistent. In order to make use of past dynamic context information an additional value-added service would have to be integrated to the federation layer which stores filtered context to a persistent data store.

Discovery of Context Resources The Area Service Register manages all necessary information for querying the local context models and discovering services. Since this register itself is not distributed, it represents a possible bottleneck for the Nexus platform whenever highly scaled simultaneous access to the register occurs. It is also a critical component with respect to availability. If the Area Service Register fails Nexus will fail.

Security and Privacy This key factor with respect to the acceptance of a context management system is yet not supported by the Nexus platform though different approaches have been investigated by now. For example, the identity of users could be veiled in order to prevent the assignment of contextual information to particular persons. Also the detail of the spatial information could be decreased if users demand data access though they are not authorized.

3.2. CoBrA

At the University of Maryland in Baltimore Country (UMBC) the *Context Broker Architecture* (CoBrA) is being developed. It represents an ontology-based universal context management framework, and describes itself to be a "[...] *new pervasive context-aware computing infrastructure* [...] *to support ubiquitous agents, services and devices to behave intelligently in according to their situational contexts*" [CF03].

Special interest in CoBrA's design is dedicated to the interoperability with intelligent artificial agents, which represent reusable context-aware software systems maintaining the state of context in local areas. CoBrA is aimed at providing a context management architecture for smart agents which can autonomously reconfigure themselves to support individual aware applications. The on-line encyclopedia "Wikipedia" provides the following definition for software agents[1].

Definition 8: Software Agent

*A **software agent** is a piece of autonomous, or semi-autonomous proactive and reactive, computer software.*

The contextual information acquired from the user's environment is aggregated into ontologies as a common contextual information pool, which supports context reasoning and sharing, as well as data protection according to security and privacy concerns. With respect to the five tiers of ontology, which have been presented in chapter 2.4.2 (see table 2) the software agents correspond to the bottom tier.

Since those agents are often operating on mobile computing devices with limited computation resources, CoBrA combines all key tasks of context management on behalf of the ubiquitous agents into a centralized approach regarding the framework's architecture.

3.2.1. Architecture

As depicted in figure 19 the central component of the architecture is the context broker. For each application domain there is exactly one broker which is pre-configured with respect to available agents.

The architecture of CoBrA is based on Java, and in particular the JADE[2] application programming interface (JADE API) which implements the FIPA[3] specifications. These specifications define a programming language-independent interoperability between agent-based software applications. Since a complete description of the FIPA specifications is beyond the scope of this thesis, the interested reader is referred to the literature resources which are

1 http://en.wikipedia.org/wiki/Software_agents
2 The **J**ava **A**gent **De**velopment **F**ramework is a FIPA-compliant API for developing multi-agent software systems. http://jade.tilab.com/
3 **F**oundation of **I**ntelligent **P**hysical **A**gents

available at the FIPA website[1].

Based on the key requirements for context management the context broker is responsible for the context acquisition and aggregation, context reasoning and inconsistency processing, context sharing to enable teamwork coordination between context-aware agents, persistence management, security and finally control of the user's privacy.

To achieve context-awareness, the context broker employs a service-oriented communication infrastructure to access static contextual information from external sources like information servers, Web resources specified in RDF[2], DAML+OIL[3] and OWL[4], or also by querying database management systems.

Dynamic context can be retrieved from sensor systems in smart environments and presented on context-aware devices using the XML-based Simple Object Access Protocol (SOAP). Context-aware agents are able to connect to the broker via the FIPA agent communication language (ACL).

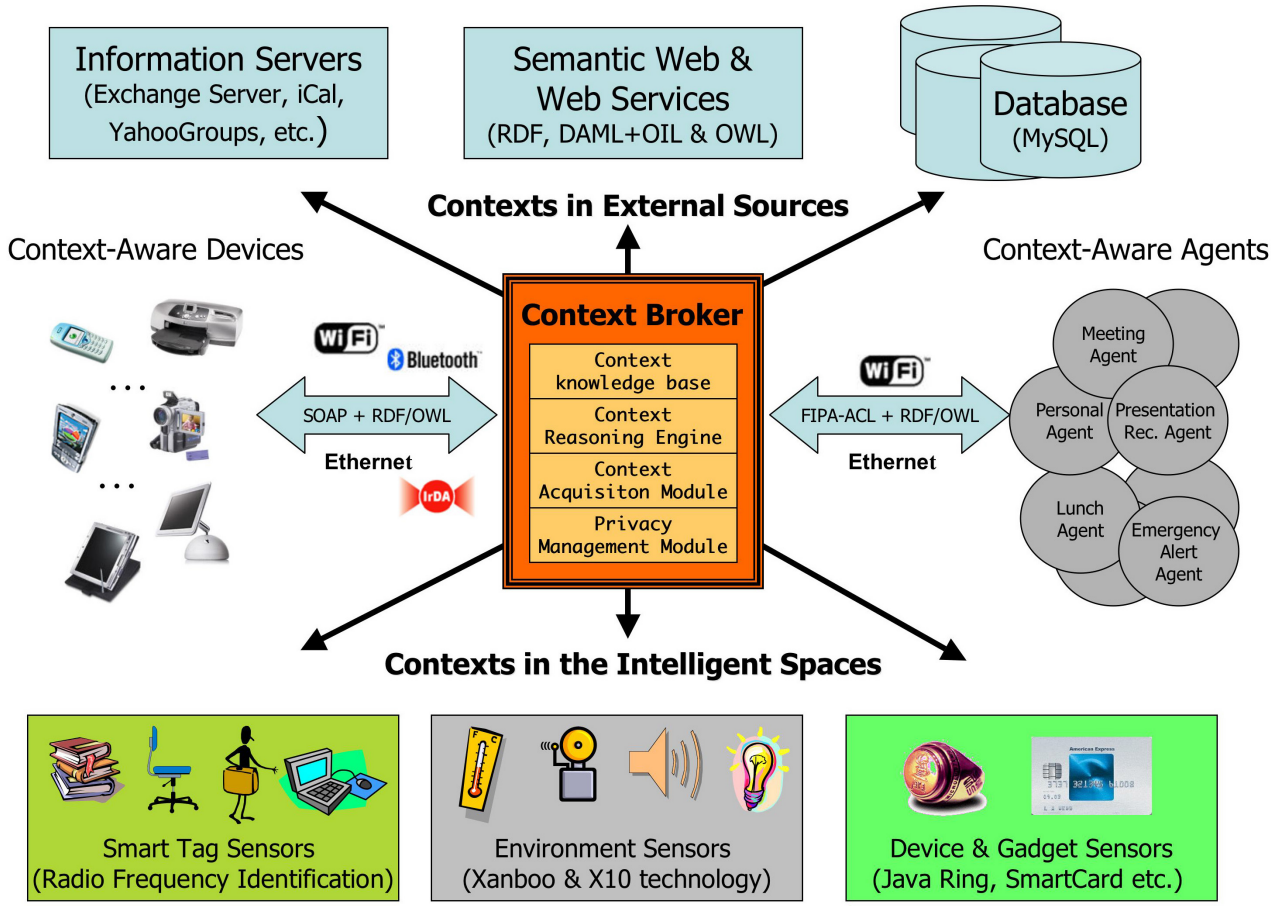


Figure 19: A survey of the Context Broker Architecture (CoBrA) according to [CHE03a]

1 http://www.fipa.org
2 The **R**esource **D**escription **F**ramework is a XML-based language used for describing and representing Web resources.
3 The **D**ARPA **A**gent **M**arkup **L**anguage and the **O**ntology **I**nference **L**ayer are XML-based languages which are based on RDF and can be used to specify ontological knowledge.
4 The **W**eb **O**ntology **L**anguage is derived from DAML+OIL

The current implementation of the context broker consist of four modules which are shown in figure 20 and are explained as follows.

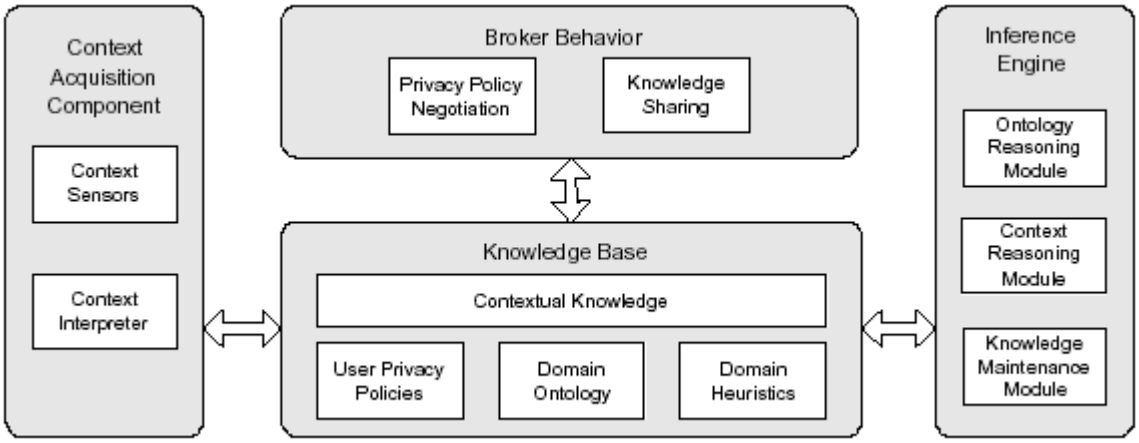


Figure 20: The conceptual design of the CoBrA context broker according to [CHE03b]

Context Acquisition Component To abstract the direct access to low-level sensor systems this component serves as a two-layered middleware for the context acquisition. The first layer declares *context sensors*, which represent both physical sensor devices and virtual information sources like information servers. Whereas the inference engine on the right side of figure 20 reasons about context information contained in the knowledge base and verifies contextual consistency, the *context interpreters* in this module pre-interpret context data which is directly acquired from the context sensors. The context interpreters are based on the domain ontology of the knowledge base and also store their interpretation results to the knowledge base.

Knowledge Base This module aggregates all contextual information of the CoBrA framework in OWL ontologies and stores the data in a relational database[1]. It first contains the ontologies which describe *contextual knowledge* about the application domain. Furthermore it stores user privacy policies which represent rules regarding the sharing of context data between agents and users. Each rule corresponds to certain types of information. It is specific for an individual user and a set of agents.

Furthermore the *domain ontology* represents the necessary information about...

- which types of context sources can be accessed by the broker,
- how it should acquire and reason about contextual information,
- what agents are accessible and what services do they provide, and
- which general types of context information are sensitive and are therefore an issue to the privacy management component.

1 Currently MySQL is used as the underlying database management system.

Finally the context knowledge base also includes so-called *domain heuristics*. These are specified by rules enabling the broker to incorporate a global shared context representation and to disintegrate contextual inconsistencies.

Inference Engine The rule-based inference engine of the CoBrA's broker is dependent on context information of the knowledge base. Three different modules are applied for different kinds of context interpretation.

The *ontology reasoning module* is based on the domain ontology within the knowledge base. It uses interpretation rules defined in the domain ontology in combination with description logic to only deduce static and terminological context information. This is achieved by applying contextual class relationships (symmetry, transitivity, etc.). Furthermore the ontology reasoning module interprets information about available services and agents. Also privacy concerns can be interpreted on basis of the user privacy policies specified in the domain ontology.

For example, if the domain ontology contains a privacy rule that classifies all context acquired in a particular office as sensitive, then all information which is inferred from context in this location is also rated sensitive.

The *context reasoning module* is intended to interpret dynamic context information of entities within the real world. As stated in [CHE03b]: "*The underlying reasoning mechanism of this module may be a hybrid composition of logic reasoning (e.g. deduction, fuzzy logic etc.) and statistical analysis (e.g. decision trees, Bayesian networks etc.)* [...]".

The third component of the context inference engine is the *knowledge maintenance module*. This module is purely dedicated to guarantee the consistency of the stored contextual information. Inconsistencies may occur due to noisy sensor data or because of an asynchronism between the observed reality and the physical world.

Broker Behavior This module controls the interaction with users and artificial agents on basis of policies. The behavior is specified in procedure protocols which are acknowledged by the context broker.

The *privacy policy negotiation* is a procedure which is initiated if a software agent intends to access context information about a user for whom no privacy rules have been defined yet. In this case based on a default privacy policy the user is asked to agree, reject or modify the offered privacy rules. The querying software agent is granted access to the user's context data only if the user agrees to the negotiated privacy policy.

Knowledge sharing is the second component in the broker behavior module. Based on the privacy policies of users their context information can be shared among particular agents using three different logical protocols which are based on the FIPA ACL specification.

- ***inform protocol*** This protocol allows an agent to inform the broker about contextual information. The broker may accept or disagree with the information depending on the logical consistency of the offered context data.

- ***query protocol*** If an agent requests context information from the context broker, the broker will either refuse to supply the data if the privacy rules do not permit the agent to access it, or inform the agent about the queried information if available.
- ***subscription protocol*** Depending on the privacy policy an agent can subscribe to the context broker to be notified about contextual changes by means of an event-based communication mechanism.

3.2.2. Discussion

CoBrA represents a centralized approach of a context management framework. The broker in the middle of the architecture implements all requirements for handling general contextual information. The architecture of the context broker involves common advantages but also considerable limitations.

Benefits and Limitations of the Broker-centric Architectural Style

The principal *benefits* of the centralized component-based broker architecture are the facilitated managing of the components' software complexity and the gain of time when developing, testing and debugging the broker modules. The division of the context management tasks into four components with each containing sub-components clearly separates the concerns of the context management requirements. Furthermore it is possible to exchange individual modules or add additional components to the broker to increase its functionality.

The key limitation of having a centralized component – in this case the context broker – in the middle of the architecture is the single point of failure. It means, that if the broker becomes unavailable, then the complete context-aware system including all dependent agents will fail. In [CHE04] Chen proposes the *persistent team* approach to solve this problem.

This approach represents a replication strategy for context brokers and ensures their availability. In [FK02] the authors provide the following definition of the term “replication”, which is translated from German.

> **Definition 9: Replication**
>
> *We speak of the **replication** of a relation R, if an instance of R is stored at two or more nodes in a network.* [...] *Dependent on the respective replication strategy all instances are updated within a limited period of time after a modification to one or more of the instances occurred.*

The idea behind this approach is that brokers are always operating in groups with a pre-defined number of members, that share a common context knowledge and are continuously syn-

chronized. If brokers fail in a group so that the number of remaining group members drops below a specified threshold, then new broker processes are executed until the initial group size is reached again.

Though replication increases the availability, scalability and parallelism, it is often not applicable due to the immense costs for synchronization after updates. So-called lazy replication models provide a possible solutions to this problem since they enable a late propagation of updates. On the other hand they would imply short-term inconsistency of contextual information of the replicated brokers.

Other disadvantages address the component-based architectural style of the context broker. Limitations are encountered when unfamiliar developers have to learn the concepts and infrastructure of the complexity of the broker's components for the first time. Additionally the amount of time (and the amount of money in commercial utilization), which is needed for integrating new components and for maintaining the system and the component interfaces, is rather high.

I would also like to briefly discuss the service-oriented communication architecture which is used for accessing contextual information from external sources, context-aware devices and smart software agents. It allows for a loosely coupling of communicating platform-independent components over public interfaces that are based on open standards. For example, using SOAP as a high-level message delivery protocol or the FIPA agent communication language enables the abstraction of the message delivery of contextual information into their constituents – context data and the transport mechanism.

The disadvantage is that the technology is yet immature. Different implementations of the standardized specifications exist which not always guarantee the interoperability, transaction paradigm, quality of service, confidence and security.

Context Acquisition and Aggregation As explained earlier CoBrA is able to acquire information from heterogeneous context sources. The context data is aggregated into an ontological knowledge base which provides a powerful mechanism for sharing its information among intelligent agents. However CoBrA suffers from the limitations of context management frameworks which are purely based on contextual ontologies (see chapter 2.4.2). The key disadvantage is the inefficiency when confronted with highly detailed and dynamic spatial context.

Context Interpretation and Disintegration of Contextual Inconsistencies In addition to the pre-interpretation of acquired context in the context acquisition component the inference engine of the broker allows for ontological context reasoning about terminological information based on description logic and reasoning about situational dynamic context with the help of user-defined rules. Also contextual inconsistencies contained in the ontological knowledge base can be resolved autonomously.

Persistence Management In principal using ontologies for modeling common concepts and dynamic situational information enables the storing and retrieving of context data by means of Web resources and database management systems.

However the persistent knowledge base of the context broker uses a relational database system. Though these databases are very efficient with respect to dealing with high amounts of data items, the mapping of ontology information to relational database schemes is rather costly.

Discovery of Context Resources Currently, available context sensor systems have to be specified to the framework explicitly. The dynamic discovery and configuration of sensors is a topic to future works, and may, for example, utilize the UDDI[1] protocol to find (Web) services, which can then be contacted via SOAP. Agents could find a context broker in range using the FIPA protocols. This approach would require that the context broker itself is implemented as a FIPA agent.

Security and Privacy Special interest has been dedicated to ensuring *security* of context information and the user's *privacy*. The broker behavior component includes modules for establishing access control to classified information which is identified by user *privacy* policies.

3.3. Context Toolkit

The *Context Toolkit* is developed by the Ubiquitous Computing Research Group at the Georgia Institute of Technology in Atlanta. As another context management framework the context toolkit neither follows the coordinate-based approach of conventional context models as it is done in Nexus, nor addresses an ontology-based architecture like CoBrA does. In contrast it represents a toolkit approach which makes massive use of context abstractions.

To facilitate the development of context-aware applications, the Context Toolkit clearly distinguishes between the acquisition of context and its usage. This allows the developers of aware applications to concentrate on the application's domain without having to implement the extensive mechanisms for collecting context information themselves [EBDN03]. The responsibilities of acquiring, collecting and managing context are resolved into five abstractions of a conceptual architecture which can be designed as reusable and customizable tool components.

In [DSA01] Dey et al. propose a conceptual architecture for context management, which is presented in the following section. The Context Toolkit architecture as a concrete implementation of these concepts is then provided afterwards.

1 **U**niversal **D**escription, **D**iscovery and **I**ntegration

3.3.1. Conceptual Architecture

According to [DSA01] the conceptual architecture for a toolkit approach of a context management framework consists of distributed context widgets, aggregators, interpreters, services and one or more resource discoverers. The control and data flow between these components is dependent on the actual implementation. Figure 21 depicts an example setup for this conceptual architecture whose common components are explained in detail as follows.

Context Widgets As a mediating component a *context widget* is logically situated between hardware or respectively software sensors and the application, thus hiding the complexity of the particular sensor system. Besides the acquisition and history maintenance of a piece of contextual information from the user's environment (e.g. the presence of a person in a room) a context widget has to enable applications and other toolkit components to access the context data it collects. These may either directly query context widgets or subscribe to notifications which are triggered whenever the widget senses a change in the context information it is responsible for. These changes can be abstracted by the widget in order to only inform applications about significant contextual modifications in the environment.

The idea of context widgets is based on the pattern of graphical user interfaces. In [DSA01]the authors explain, that in a similar way "[...] *GUI widgets mediate between the application and the user, context widgets mediate between the application and its operating environment. As a result, just as GUI widgets insulate applications from some presentation concerns, context widgets insulate applications from context acquisition concerns.*"

Examples context widgets are responsible for determining the users' location and identity in a limited area or the temperature in a room.

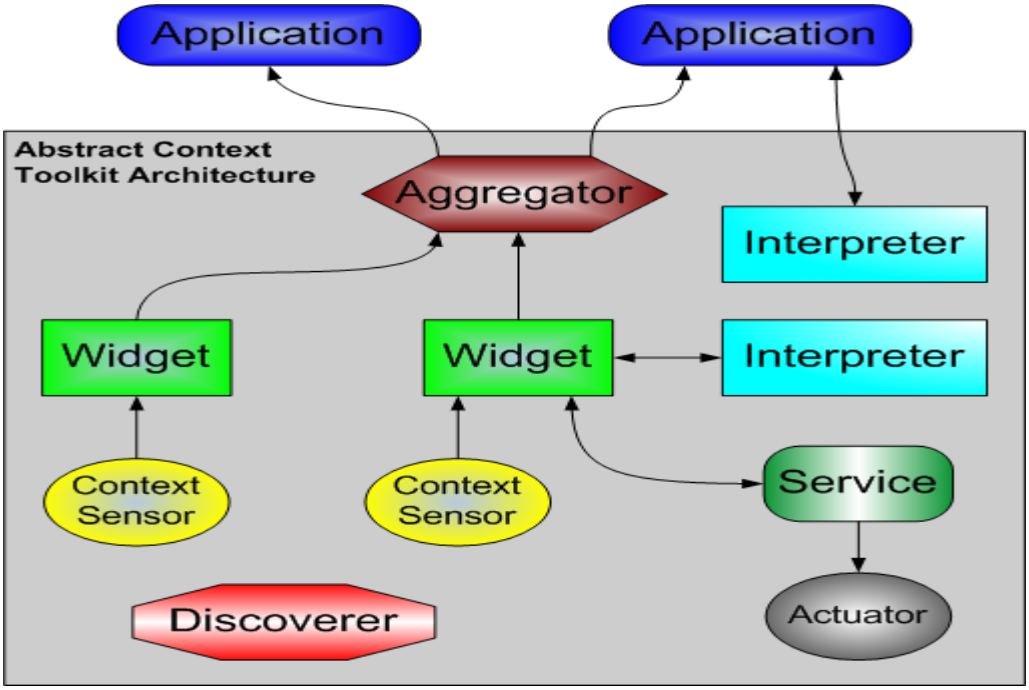


Figure 21: Example instance of an abstract context toolkit architecture with two connected applications (the arrows represent the direction of data flow)

Context Aggregators Similar to context widgets *aggregators* collect contextual data. However, the idea behind aggregators is allowing applications to access combined, coherent context information instead of having to contact several context sensors that provide this data.

In contrast to context widgets which encapsulate the handling for only a piece of context information, aggregators are responsible for incorporating multiple and often distributed pieces of context with respect to a particular entity. This context information, which is aggregated into a common representation, is also concerned to be belonging together.

For example, if a context-aware system should be able to detect whether a user is watching TV in the living room, then different pieces of context information are required before their interpretation is possible at all. At least, the system has to acquire data referring to whether the user and the TV set are located in the living room and whether the television set is turned on.

Context Interpreters These abstract toolkit components have the task of deducing higher-level context pieces from information of the various context sources. *Context interpreters* are aimed at shifting the responsibility of context reasoning from the applications towards the conceptual context architecture. Common standardized access interfaces allow the restricted reuse of interpreters in different application setups.

The restriction refers to the application specific parts of the interpreter implementation. In order to implement an interpreter, "[...] *a developer must provide the parts of the interpreter that are specific to the instance being built. These include the context attributes that are to be interpreted, the context attributes that are the result of the interpretation, and the actual code that will perform the interpretation*." [DEY00]

Context Services As the counterpart of widgets, *context services* are used to influence the context in the physical and virtual environment of users. Services can – but are not required to – use actuators to perform context-aware functions[1], which are requested by applications to output contextual knowledge in a certain way. Actuators are hardware controllers that can be used to switch on and off kitchen appliances, lights and television sets, regulate the room temperature, or output audio-visual aids to the user.

Since it is often desirable to reuse common context services and their actuators in several applications, context services should be configurable and hide the complexity of their functionality. Furthermore they can be differentiated in synchronous and asynchronous services according to the requirement of an entity's response to certain actions. A synchronous context service immediately performs a function on behalf of an application without waiting for a reaction by the entity such as a user.

If the application is dependent on an entity's response, then it has to be supposed, that the entity will not react at once. The cause for this behavior can be a user, who does not notice a displayed message box, or also an addressed external software system, which temporarily becomes unavailable. In general, the application can not delay its process until the user or an

1 These functions are described in chapter 2.2.2

external system eventually responds. In this case an asynchronous service executes the action and notifies the application when the response arrived.

Discoverers In order to provide information about the components of the context architecture, a federation of *discoverers* stores information about the presence and features of individual context widgets, aggregators, interpreters and services in the distributed environment.

On the startup of such a component, it informs a known discoverer of its presence and supplies it with its contact information and capabilities regarding context handling. Therefore other components do not have to be configured about all the individual constituents of the framework in an explicit way. Instead they can find each other using the discoverer components, that provide the same query and notification mechanism as widgets and aggregators.

Two different kinds of discovery mechanisms can be distinguished. White pages lookup refers to finding a toolkit component by indicating its known unique identifier whereas yellow pages lookup corresponds to finding a group of components that share common properties. When using yellow pages lookup the querying component has to specify characterizing attributes. For instance, an aggregator could ask a discoverer to reply all widgets which collect context information about a user in a particular room.

If a component, which is registered with a discoverer, becomes unavailable for a certain amount of time or terminates completely, the particular implementation of the discoverer has to provide a mechanism to detect this case, and delete this component from its resource discovery list. A possible mechanism is the explicit notification of a discoverer about connection timeouts, or the continuous “pinging” of all registered components.

3.3.2. Context Toolkit Architecture

The *Context Toolkit* is an implementation of the conceptual architecture described in the previous section, and is mainly written in Java.

Assumed that the employed operating systems support the TCP/IP network protocol, the Context Toolkit provides a platform-independent development of context-aware applications, that can be written in a large variety of programming languages (e.g. Java, C++, Frontier, Visual Basic, Python, PERL, Squeak). Also a lot of computer hardware devices are supported such as many mobile phones, wearable and handheld computers or embedded sensor systems in addition to conventional desktop computers and notebooks.

As depicted in the UML class diagram in figure 22, the constituents of the Context Toolkit address to the conceptual toolkit architecture except for the service class component, which is marked green. The special issue on this component is explained later in this section. The aggregator in the conceptual architecture is called a context server in the Context Toolkit. As follows, custom properties of the Context Toolkit components are presented briefly.

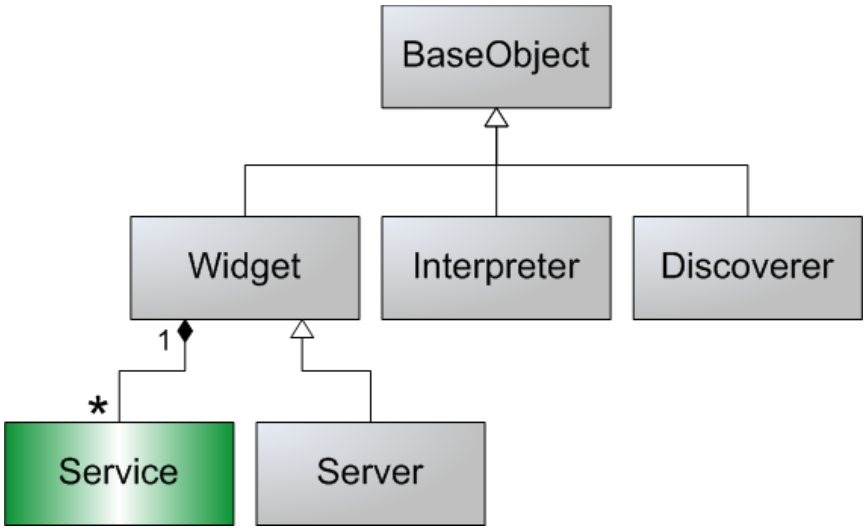


Figure 22: UML diagram of the abstract Context Toolkit classes

Base Object and Distributed Communication The Context Toolkit requires that each individual widget, interpreter, server and discoverer runs in a single process. By employing peer-to-peer communication the processes can be deployed on different network nodes and processors. To guarantee that all components are based on the same communication mechanism, they have to inherit from the “BaseObject” (the context server inherits indirectly via the widget), which provides the functionality for *distributed communication*. Though applications can subclass from “BaseObject”, they may also just create an according instance to be able to communicate with the toolkit components.

The distributed communication is currently based on XML-encoded messages with an uniform message syntax and HTTP as the default communication protocol based on the TCP/IP transport layer. Two types of communication are provided by the “BaseObject”.

Server-like communication enables the listening for incoming messages, which are handed over to newly created threads on arrival, thus enabling a concurrent message management. For instance, this type of communication is used with widgets in general. When the “BaseObject” (or respectively the instantiated thread) receives a message, it first tries to process it on its own. If this is not possible, it delegates the message to a handler in the concrete class, which is derived from this “BaseObject”.

Client-like communication can be used to notify another toolkit component or an application by sending a message event. The prerequisite of this event communication is the continuous availability of the destination component [DAS99].

As depicted in figure 23, the “BaseObject” employs both communication mechanisms to support an extended subscription feature, that is based on callbacks, attributes and conditions.

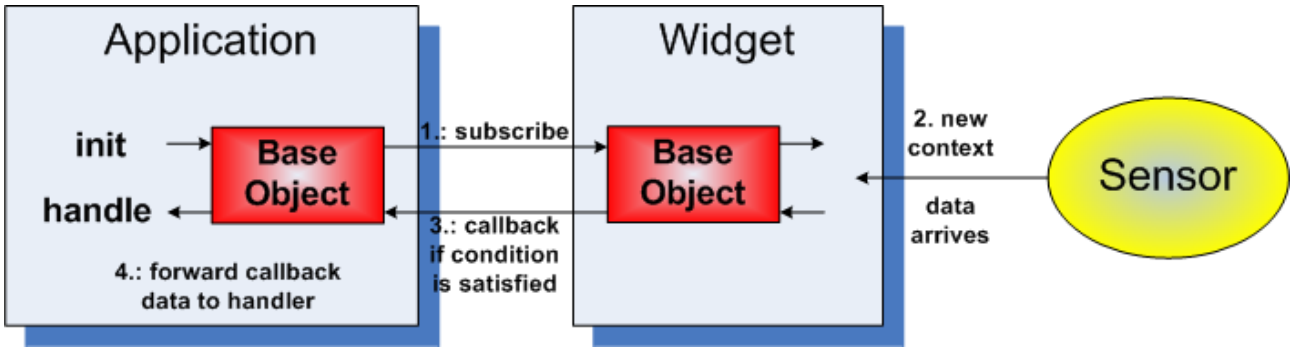


Figure 23: Subscription of an application to a Context Toolkit widget

In this example, an application subscribes to a widget to be notified about callback information, i.e. contextual events, the application is interested in. Besides the optional declaration of attributes, which correspond to the types of context to subscribe to, the extension of the subscription technique allows to specify certain conditions that have to be satisfied in order to return the callback data.

For instance, lets assume, the application is interested in the room temperature if it exceeds 24 degrees centigrade. It uses the “BaseObject” to (1.) subscribe to an appropriate widget, that senses the temperature of the according room, and that can be found with the help of the resource discovery component. In order to subscribe, the application generally has to specify its contact information, and in our example both the temperature attribute and the condition “*temperature* > *24*”. On the arrival of a new temperature value (2.), the widget first decides autonomously via custom logic whether the context change is significant. If assumed, that the temperature raised significantly from 22 to 25 degrees centigrade since the last measuring, then the widget evaluates the conditions of all its subscriptions, and will notify (3.) the application about the new temperature value using the “BaseObject” for communication. The “BaseObject” on the application side receives the callback event, and finally forwards it to the handler (4.), a dedicated object, which is responsible for only this particular type of subscription callbacks.

The subscription mechanism, which is provided by the “BaseObject”, also addresses to context servers and discoverers, and reduces the rate of capacity utilization of the distributed network since a communication is only necessary whenever changing context matches pre-defined notification conditions. Because the basic underlying communication mechanisms of the “BaseObject” are pluggable, besides the message language also the protocol can be changed. For example, the transfer protocol HTTP could be exchanged for FTP, SMTP[1], CORBA or Java RMI[2] by implementing two interfaces, one for the client- and one for the server-like communication support [DN00a].

Widgets According to the conceptual architecture *widgets* are responsible for collecting and storing pieces of context information. They offer mechanism to access context information, the types of context data (that the widget monitors changes about and that can be subscribed to), and the widget's services.

1 **S**imple **M**ail **T**ransfer **P**rotocol
2 **R**emote **M**ethod **I**nvocation

The widget represents context information by typed attribute name-value pairs where the values can again recursively define typed attribute name-value pairs. An example for this context representation is given in [DEY00] on page 49.

AttributeNameValue: Attribute with a value
Example: `[identity, string, "Anind"]`
DataObject representation:
```
{"attributenamevalue type=string",
    {{"attributeName", "identity"}
     {"attributeValue", "Anind"}}}
```

Example 1: Context representation by the Context Toolkit widgets

If not explicitly configured to turn storage off, the acquired context is automatically stored in a MySQL database table using the JDBC[1] API, where it can be accessed again if an application requests to. Context Toolkit widgets also support the caching of the last context data in main memory to increase the response performance when queried in quick succession.

Context Services The conceptual architecture declares a *context service* to be an independent component with respect to the mentioned context abstractions. The Context Toolkit allows to include none, one or more than one service into one widget. This should allow developers to think of widgets as logical units, that not only collect contextual information pieces of an environment, but also perform actions which are contextually related. Though services are encapsulated by widgets, developers can reuse previously built services, that are available in a common library.

To use a specific service within a widget, applications can query the widget for all provided synchronous and asynchronous context services. They can then demand the execution of a particular chosen service with the help of the “BaseObject”.

Context Servers The *context server* in the context toolkit corresponds to the aggregator of the conceptual architecture. As depicted in figure 22 on page 76 a context server subclasses from the widget class thus inheriting all its basic functionality including the capability for storing and retrieving persistent context information.

The server pattern follows the conceptual architecture in that it provides one server for each entity. Since the server mediates between the application and a collection of widgets, that sense related context information regarding an entity, it also incorporates all the runtime functionality of the aggregated widgets. Therefore a Context Toolkit server provides a common access interface to both context sources and actuators of context services.

Similar to widgets a server can store context information about its entity to a MySQL database table using the pluggable storage module of the widget component.

.

1 **J**ava **D**atabase **C**onnectivity

Interpreters Based on transformation rules they convert and deduce context information without maintaining an own state over individual interpretations. Any component, which supplies context information to an interpreter for inference, must use synchronous communication and wait for the interpretation result. The interpreter itself does not store the result for context reasoning but responds as soon as it is computed.

The interpreter can be accessed by other interpreters, widgets, context servers and also directly by applications. In order to develop a new interpreter, first the types of input and output context have to declared. To enable the interpreter to transform input context into output context, an implementation of the interpreter logic must be provided.

Discoverer To register with a discoverer on startup, the widget, server or interpreter has to know the discoverer's hostname or IP address. During the registration process they provide their own hostname, communication port and unique identifier to the discoverer to enable the resource discovery. Widgets and servers additionally have to supply information of the context types and services they handle, whereas interpreters inform the discoverer about the context types they can reason about and those they are able to respond as reasoning results.

In addition to querying for finding and locating components, discoverers provide a subscription mechanism for informing applications and other Context Toolkit modules about the detection of new or the removal of terminated context components. A component, which stops execution, notifies the discoverer about its unavailability. In the case, that a toolkit module crashes, the discoverer will detect this after a configurable number of faulty attempts of “pinging” this component.

As stated in [DEY00] on page 57 the Context Toolkit architecture “[...] *allows for multiple Discoverers.* [...] *Components are not limited to registering with a single Discoverer and can register with multiple Discoverers* [...]”. These can also be organized in a federation hierarchy which establishes a common information pool for the Context Toolkit components' contact information and capabilities.

3.3.3. Discussion

The toolkit approach of this framework, which is based on the presented conceptual architecture, is aimed at separating the concerns between the acquisition, aggregation and usage of contextual information. However, besides its desirable benefits of context management it also involves limitations, which impair its usability in developing context-aware applications.

Benefits and Limitations of the Context Toolkit Architecture The Context Toolkit should enable the rapid prototyping of applications by reusing certain components. However the reusability is restricted because of the tight coupling with application logic. Widgets have to be adapted to decide what context information represents a significant contextual change. They also incorporate particular services that have to be exchanged when building new applications. These topics refer to context servers, too, which are actually extended widgets. Also

interpreters can often not be reused since they generally implement the custom reasoning capabilities for a certain application domain.

In another aspect both applications and the toolkit modules can reuse uniform distributed communication mechanisms, which are encapsulated into the “BaseObject”. However an interaction is only possible between two communication partners at the same time. The employment of XML-based messages delivered via HTTP is rather inefficient. Messages, which are represented in XML are verbose, contain a lot of redundant information, which are not required to be transfered, and involve complex parsing operations. HTTP transmits data only in plain character text and does not guarantee a constant communication rate. Also the Context Toolkit only enables the transmission of discrete information. Therefore streaming data like videos or sound can not be handled easily. Instead of transferring the complete continuous information at once, it has to be split and transmitted in discrete frames.

Every component of the Context Toolkit runs in a separate process. On the one hand this allows a very good distribution over different network nodes and processors. On the other hand it restricts the framework's scalability due to the many communication channels and changes of concerns between different processes.

Context Acquisition and Aggregation Both *acquisition* and *aggregation* of context information is provided in a custom way by the Context Toolkit. The context is represented in form of typed attribute name-value pairs [DN00b]. This representation is sufficient for describing basic low-level context, but impairs the knowledge declaration of complex contextual relationships, and does not allow for a declarative terminological knowledge representation. In contrast to a formal context modeling, which has the aim of knowledge sharing, and provides automated adaption and modification of information by other context management frameworks, the Context Toolkit uses an informal and therefore not standardized context representation.

Context widgets incorporate services providing means of influencing the contextual environment with the help of actuators. Though the combination of both toolkit components allows developers to regard a context widget as a uniform entity for accessing parts of the real world, it affects the reusability of implemented instances of both context abstractions (widget and service) by its tight coupling.

Context servers aggregate context information from different widgets and support the mediation between them and applications. Therefore applications are not forced to maintain contact information of all their required widgets on their own.

Context Interpretation and Disintegration of Contextual Inconsistencies The *context interpretation* provided by the Context Toolkit is a rule-based transformation of input into output data. It does not support the reasoning about terminological concepts or common relationships.

Another concern is the tight coupling of implementation code of interpreters and application domain logic. Parts of the interpreters have to be implemented and adapted to the target domain whenever new applications are developed.

Due to the synchronous communication interpreters block the program run of calling components. That can be unacceptable for a costly inference processing. On the other hand the composition of connected interpreters corresponds to the pipes and filters architecture, and also benefits from its advantages. Each interpreter refers to a filter, that processes information, and each pipe refers to the communication capabilities of the “BaseObject”, which connects the interpreters. Filters or interpreters can be easily modified, exchanged and combined in a different ordering. The separation of complex interpretation tasks into logical small processing units, that can often compute different jobs in parallel, improves the efficiency in distributed environments. However, since interpreters do not maintain any state information, the debugging and error handling is difficult, if many interpreters are connected together.

Disintegration of contextual inconsistencies and *ambiguities* is not supported by the Context Toolkit. The context management of the toolkit is based on the assumption, that the context information is 100 percent accurate, which is not possible if sensor data is sourced from hardware sensors. In general these acquire almost always ambiguous context information.

Persistence Management *Persistence management* of context data is provided by both widgets and context servers in form of a JDBC-based connection to a possibly remote relational database management system (DBMS). Though currently MySQL performs this task, it can be exchanged for another relational DBMS. This is possible by means of the standardized JDBC API applying to four different types of JDBC drivers, with which almost any relational database can be accessed [HCF97].

A disadvantage is the duplicate storage of context information. First, context data is stored to a relational database table by widgets after its acquisition from sensors. When the widgets notify a context server about significant changes of context, the server additionally stores the aggregated context into another table. However, it is possible to configure each single widget and server whether they should store context or not.

Discovery of Context Resources Though the discoverer and the federation of discoverers within the Context Toolkit have already been announced in 2000 [DEY00], the discoverer component was presented as an implemented module just recently. Unfortunately their is no detailed implementation documentation about this component at presence.

Using the white or yellow pages lookup capability of the *discoverer,* applications and Context Toolkit components should be provided with detailed information about all the toolkit modules they are interested in. The subscription mechanism should be employed to inform a subscriber about any changes with respect to the availability of required components.

Security and Privacy Except for the transmission and verification of an unique authentica-

tion identifier when accessing historical context information, currently the Context Toolkit does not support a *security* or *privacy* mechanism. Even if not any intruder had the chance to easily connect to all components of the Context Toolkit infrastructure to spy out sensitive context information, the plain character text transmission via HTTP can also be easily tapped.

3.4. Summary and Conclusion

In the previous sections three different context management systems were investigated, which – in my opinion – are the most promising frameworks at presence.

Though they are well-suited for respective particular kinds of context-aware applications, none of the presented architectures combine the requirement of managing a high dynamic complexity of spatial context abstractions in a large spatial scope with the possibilities of contextual ontologies.

Nexus The layered architecture of the *Nexus* platform is well-suited for a conventional coordinate-based management of spatial context. However it does not incorporate terminological concept modeling and reasoning. The interpretation of context information that regard both moving or static entities is also the responsibility of the applications.

Though all other parts of the *Nexus* platform are distributed, the *Nexus* Area Service Register (the resource discoverer) is a centralized component and thus represents a single point of failure and a possible bottleneck. With respect to security and privacy concerns up to now the continuous development and the implementation of *Nexus* components did not address this topic.

The creation of new applications on top of the *Nexus* platform is also rather complicated, because their communication subsystem has to be developed from scratch in order to connect to *Nexus*. Currently work is in progress with high effort dedicated to building a support framework, which encapsulates basic services facilitating the communication with and usage of the *Nexus* platform [WIE05].

CoBrA The component-based centralized architecture of the context broker, that is the main module in the framework does not satisfy the requirement of a distribution of all context management tasks. Though context brokers could be joined together via a replication strategy, in my opinion the distribution should be applied on a rather low level as it is done in the Context Toolkit. Instead of having a number of large-grained context brokers located on computers in the user's environment, the distribution of small-grained context widgets and some medium-grained aggregators (context servers) requires considerably less computing resources. These can be employed on a large variety of wearable computer devices such as mobile phones, handheld computers and others, which do not provide much computing power and main memory.

With respect to the management of high dynamic spatial information, the ontology-based con-

text handling is clearly not efficient enough. *CoBrA* also does not support the dynamic discovery of context sensors at present.

Context Toolkit Though the toolkit-based conceptual architecture presented in chapter 3.3.1 specifies a successful abstraction of context management responsibilities, the concrete implementation does not completely conform to it. The main limitations refer to the inappropriate tight coupling of common context management tasks with application logic and the limited scalability. Also the disadvantage, that a constant communication rate can not be guaranteed by the distributed communication mechanism of the “BaseObject” restricts the employment of the *Context Toolkit* with applications, that need to be informed about high dynamic context information.

The context representation by means of attribute name-value pairs is not very suitable to model detailed locational context information. In this case a coordinate-based approach is desirable. Also the inability to terminologically model common concepts of the real world requires to implement context interpreters from scratch whenever an application is created in an unrelated domain. Another concern is the blocking behavior of the context interpreters. Assumed, a lot of complex and dynamic spatial context information has to be reasoned about using costly inferences, the possible frequent interruptions of the program flow do not address the real-time requirements of Augmented Reality systems.

The *Context Toolkit* does also not provide a mechanism to resolve logical inconsistencies and also contextual ambiguities, which can not be avoided when working with naturally imperfect hardware sensors as observing tools of the real world.

Finally the almost missing of access control and privacy management in the *Context Toolkit* will not increase the acceptance of its context-aware applications by the public.

Conclusion In principal the limitations of the presented context management frameworks are grounded on the disadvantages of the single employment of either coordinate-based context models (Nexus), ontology-based context representation (CoBrA) or a toolkit approach instead (Context Toolkit). None of the frameworks is really well-suited for managing highly dynamic *implicit* spatial context or handle abstracted spatial queries.

Therefore I decided to develop an implementation of a combined approach of conventional context models and contextual ontologies, which is already introduced in chapter 2.4.3.

4. Overview of Technologies for Supporting Ontologies

This chapter is dedicated to provide an overview of technologies, which can be used to model concepts of the augmented world. Because currently there are no ontology-based approaches, which are specialized in the handling of spatial context information, general languages and reasoners are investigated, which can be restricted respectively used for this purpose.

4.1. Languages

In chapter 2.4.2 contextual ontologies and their capabilities of supporting the knowledge modeling for aware systems have been presented. Since the specification of ontologies in nonuniform representations contradicts to the idea of knowledge sharing and information reuse, ontologies should be describes using common languages. The idea of employing ontologies also in context management is originated from research efforts in the next generation Web – the “Semantic Web”.

Semantic Web The *Semantic Web* is a rather new area of research in computer science. An article in the Scientific American in May 2001 explains that it “[...] *is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*" [BHL01]. The principal idea behind the *Semantic Web* is the enrichment of syntactic information by semantics in order to provide the foundation to computers that they not only store or present data to humans, but also “understand” its meaning and be able to reason about it [DFH02].

The foundation of the *Semantic Web* is the description of terminological knowledge about the information which has to be processed in dependence of its context. To enable computers to uniformly integrate ontologies into their information processing and interpretation tasks, a *standardized* representation format is required. This format comes with RDF(S) and OWL.

The following sections will discuss the languages RDF(S) and the OWL language family, which are suitable for describing ontological information. They are recommended by the W3C, and are based on XML in principal. Since the explanation of each of these languages would fill whole textbooks, I confine myself to describe only the relevant parts, that are related to the modeling of ontologies. Also the ontology languages DAML, OIL and DAML+OIL are only mentioned in the section about OWL due to their great similarity and because of the fact, that OWL is originated from them. However references are provided for the interested reader. Here it is also assumed that the reader is familiar with the concepts of XML and XML Schema, that are explained in [CHA02] and [VLI02] for instance.

The most important advantages of using XML for specifying languages are its flexibility and simplicity of application. It is legible by both humans and computers, and enables software compatibility and portability by self-descriptive information combined with a universal charac-

ter encoding. Currently there are also numerous tools available, that facilitate creating, parsing and storing XML documents.

Evaluation Criteria First the presented languages are investigated for their expressiveness and the support of sharing the specified knowledge between independent context management systems, which use the same language. Additionally they must support the automated deduction of ontological information, and enable the reuse of previously declared ontologies in new applications.

4.1.1. RDF and RDF Schema

The Resource Description Framework[1] has been recommended by the W3C in 1999. It specifies a simple data model for describing resources, and is intended to add meta-data to them in order to improve the discovery of information by autonomous software agents.

In the context of RDF a resource is anything which can be addressed by an URI[2] and is therefore anything which is accessible via the Web, such as a Web site, one of its sections, its author, hers children, theirs school books, etc. In this context RDF can be used as a *basis* to describe ontological information as Web resources.

As mentioned before, the RDF model is simple. It consists of one or more RDF statements, each of them represented as a triple of subject, object and predicate which form a directed graph with the subject and object as its nodes, and the predicate as the edge between them and always directed towards the object. The statement's subject is a resource, each of its predicates denotes a subject's property, which is also identified by an URI reference, and the object is either a resource or a literal.

Figure 24 shows an example data model. Here it can also be seen that RDF allows for *reification*, i.e. the nesting of statements so that a statement can be the subject of another statement.

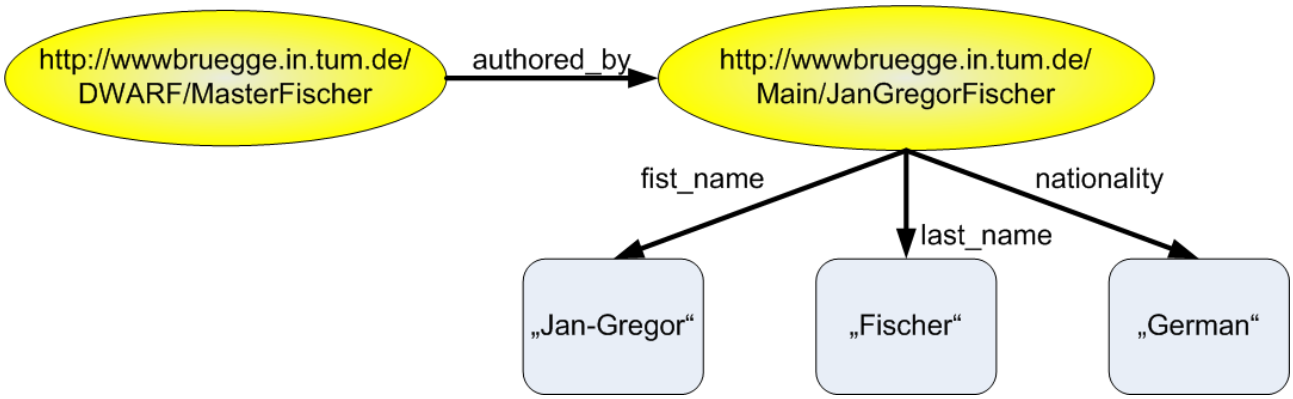


Figure 24: Example RDF model (the predicates are also implicitly RDF URI references)

1 http://www.w3.org/RDF/
2 **U**niform **R**esource **I**dentifier

To denote that a resource is an instance of a category of resources, RDF provides the predefined property “type”, which maps the subject referred to as an instance to the object considered as the class or category the subject belongs to. It is possible to list resources and literals in three kinds of unbounded containers. Whereas a *bag* contains unordered, a *sequence* holds ordered elements. There is also a collection type, which lists alternatives of elements.

However RDF itself does not provide sufficient means for a formalized description of common relationships between classes of resources. Therefore RDF Schema has been recommended by the W3C for the specification of RDF vocabularies.

RDF Schema

RDF Schema[1] uses RDF resources to provide a standardized extension to RDF. It can be used to declare semantic information about RDF statements in order to restrict their utilization to a common vocabulary when describing ontologies. This vocabulary of RDFS itself is assigned to a namespace, which is denoted “rdfs”[2] here. Since RDFS extends RDF, the corresponding namespace referred to as “rdf”[3] is also used in the RDFS specification.

The schema specifies a type system of hierarchical structured classes and properties, which resembles to object-oriented languages. However in this case the properties of class instances are not specified inside a class, but as individual properties with a resource domain and a range. The following figure depicts the core language constructs, which are available in RDFS [W3C04d].

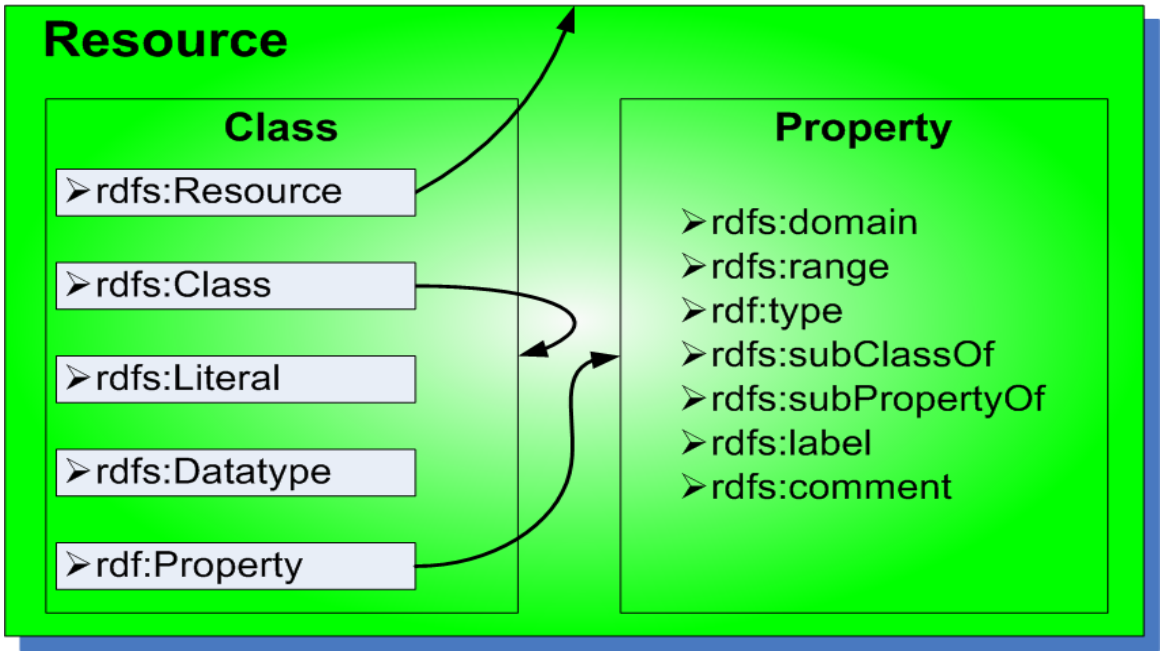


Figure 25: Core vocabulary of RDFS

1 http://www.w3.org/TR/rdf-schema/
2 XML namespace rdfs: "http://www.w3.org/2000/01/rdf-schema#"
3 XML namespace rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns#"

Core Classes In contrast to RDF the schema enables a separation between the ontological terminology and the corresponding instances. Thus it is possible to model common types of resources. Unfortunately it is rather complicated to read RDFS because of the involved recursions above all between “rdfs:resource” and “rdfs:class”. Resources can be categorized into common *classes*, where each class individual is again a resource. As mentioned before when describing RDF, everything which is stated in RDF is a resource, and now is an instance of the RDFS class “rdfs:Resource”. That means they have the implicit property “rdf:type” with the RDF statement's object “rdfs:Resource”.

```
<rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Resource">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Resource</rdfs:label>
  <rdfs:comment>The class resource, everything.</rdfs:comment>
</rdfs:Class>
```

Example 2: Description of the class "rdfs:Resource" in RDF/XML [W3C04d]

“Resource”, “Class”, “Literal”, “Datatype” in the namespace of RDFS and “rdf:Property” are all instances of “Class” (also “rdfs:Class” is an instance of itself). It denotes the class of resources which are RDF classes. The class “literal” specifies literal values such as boolean, integer, double or string values. If a value is typed then it is an instance of “rdfs:Datatype”. Finally “rdf:Property” denotes the class of RDF properties.

Core Properties *RDFS properties* describe a relation between subjects and objects. The constraints “domain” and “range” can be applied to a property in order to restrict its domain of applicable resources and its range of resources representing the result. The property “type” with a domain “R” and a range “C” denotes that the resource “R” is an instance of the class “C”. The “subclassOf” and “subPropertyOf” properties allow for class, respectively property inheritance. Finally the “label” and “comment” properties can be used to represent the name of the resource and a descriptive comment in human-readable form.

Example 3 on the next page demonstrates how ontologies can be represented in RDFS. Here three entity classes and two properties are modeled.

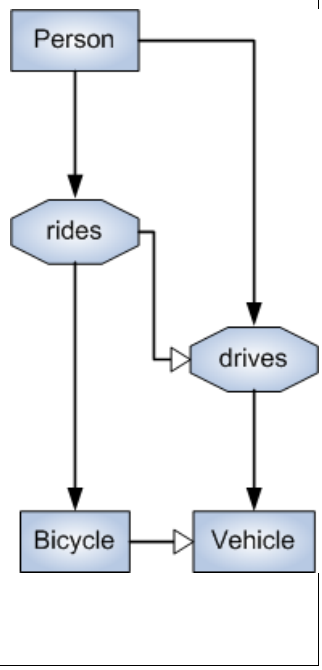
```
<rdfs:Class rdf:about="http://.../terms#Person">
  <rdfs:label>Person</rdfs:label>
</rdfs:Class>

<rdfs:Class rdf:about="http://.../terms#Vehicle">
  <rdfs:label>Vehicle</rdfs:label>
</rdfs:Class>

<rdfs:Class rdf:about="http://.../terms#Bicycle">
  <rdfs:label>Bicycle</rdfs:label>
  <rdfs:comment>Bicycle subclasses Vehicle</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://.../terms#Vehicle"/>
</rdfs:Class>

<rdf:Property rdf:about="http://.../terms#drives">
  <rdfs:label>driving a vehicle</rdfs:label>
  <rdfs:domain rdf:resource="http://.../terms#Person"/>
  <rdfs:range  rdf:resource="http://.../terms#Vehicle"/>
</rdf:Property>

<rdf:Property rdf:about="http://.../terms#rides_bicycle">
  <rdfs:label>riding a bicycle</rdfs:label>
  <rdfs:domain rdf:resource="http://.../terms#Person"/>
  <rdfs:range  rdf:resource="http://.../terms#Bicycle"/>
  <rdfs:subPropertyOf rdf:resource="http://.../terms#drives"/>
</rdf:Property>
```



Example 3: RDFS class and property inheritance in RDF/XML

A more detailed discussion of the resource description framework and the schema would go too far at this point. However the interested reader is referred to [W3C04d], [W3C04e] and [SEMD00] for more information about RDFS, RDF and the modeling of ontologies with these languages.

Discussion RDF itself is not suited for modeling ontologies, because it completely misses restrictions of the semantics of predicates. These basic requirements for describing ontological information first come with RDFS. It allows for a hierarchical representation of classes and properties with the help of the property constraints “domain” and “range”.

However the expressiveness of RDFS does not include comprehensive formalized language constructs to relate classes, properties and instances by means of (in)equality, union, intersection or complement specification. As a matter of fact, it is hard to incorporate an ontology with others describing similar knowledge, that could be integrated with the help of these operators.

Besides the missing of cardinality constraints there is also no formalized specification with respect to types of properties. For the interpretation by inference systems it would be helpful to additionally describe the relationships of properties (e.g. symmetric, transitive or inverse).

4.1.2. The OWL Language Family

The acronym stands for “Web Ontology Language”[1] and represents a family of languages, that has been recommended by the W3C in 2004 [W3C04b] for describing common concept knowledge not only for the Web, but also for various other fields of employment in computer science. Possible example cases for its usage are given in [W3C04c]. Web portals can make use of OWL to increase the capabilities of indexing and finding information content. Multimedia collections can be enriched with additional semantics about videos, songs and pictures. Web services and agents could be based on OWL ontologies to combine peoples' interest with content from various information sources. In ubiquitous computing the ontology language can be used to define the interoperability for computer devices and services.

Considering the example of CoBrA (chapter 3.2), it could be seen that OWL ontologies also support the requirements of context management and context-aware applications. Because of its great variety of employment, OWL can be considered for the reasoning about spatial context, which is accessed by Augmented Reality applications.

The Roots of OWL OWL is originated from the experiences of the combined research efforts in the ontology description languages DAML[2] and OIL[3]. The research center DARPA (Defense Advanced Research Projects Agency) of the Department of Defense of the United States has officially initiated the DAML project in August 2000 in order to investigate approaches to knowledge representations for the World Wide Web of the next generation, i.e. the Semantic Web. About the same time a similar project has been started in Europe, which had the same objectives.

The cooperation of both research groups lead to the combination of both languages into “DAML+OIL”[4]. Finally OWL follows from the lessons learned of DAML and OIL, and is based on RDF and RDF Schema just as its predecessors.

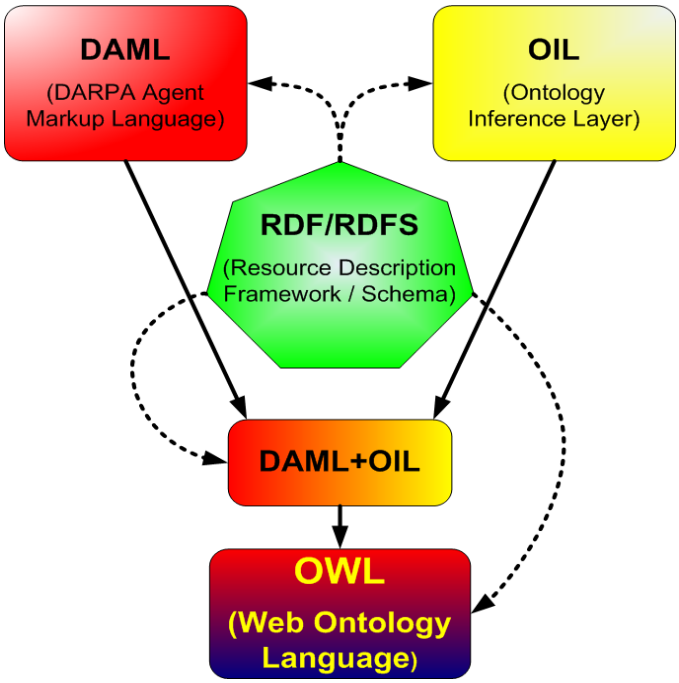


Figure 26: The roots of OWL

1 Not “Ontology Web Language” as assumed when comparing the abbreviation to the capital letters.
2 http://www.daml.org/
3 http://www.ontoknowledge.org/oil/index.shtml
4 http://www.w3.org/TR/daml+oil-walkthru/

4.1.2.1. Overview of the three Sub-Languages of OWL

The Web Ontology Language consists of the three RDF and RDFS-based sub-languages OWL Lite, OWL DL and OWL Full, which provide an increasing power of expressiveness in the order they are listed here. OWL Full extends OWL DL and OWL DL extends OWL Lite so that every legally specified ontology and every valid ontological conclusion of a weaker OWL sub-language is also a legal ontology respectively valid conclusion of the extended sub-language.

OWL Lite represents the simplest of these languages with the support for a classification hierarchy and simple properties for expressing constraints. OWL DL refers to description logics (see chapter 2.4.2.1 on page 51) and provides the formal concepts for a maximum of ontological expressiveness whilst guaranteeing both computational completeness and decidability [SCH01]. Finally OWL Full offers the free usage of the RDF, RDFS and OWL syntax including the extension of the vocabulary, and denotes the most powerful sub-language of OWL. However it does not retain computational completeness and decidability by all means. For instance, "[...] *in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.*" [W3C04b]

OWL DL and Full share the same vocabulary, which extends the OWL Lite language constructs. Whereas OWL Full violates the requirements for description logic-based reasoning systems by allowing the unrestricted combination of the vocabulary, OWL DL "[...] *puts constraints on the mixing with RDF and requires disjointness of classes, properties, individuals and data values.*" [W3C04f]

The following sections briefly describe the most important language constructs, that can be used in OWL. Since there are currently 40 different OWL-specific elements in the vocabulary (additional elements are included from RDF and RDFS), this description is not aimed to be complete. It should only provide an overview of the expressiveness of the languages. For detailed information about the comprehensive vocabulary the reader is referred to [W3C04b] and especially [W3C04f].

4.1.2.2. OWL Lite

Every document which is specified in one of the OWL sub-languages is a RDF document. Therefore it is possible to declare namespaces of useful vocabularies within the opening tag "rdf:RDF", which surrounds a XML-serialized OWL document. For instance the namespace for the XML Schema language declares data types such as numeric types, strings and date or time types, which can be used in OWL.

Header In the ontology header several RDFS tags and the OWL "import" tag can be used to describe administration information (comments and version control) about the ontology and to declare the inclusion of external ontologies.

```
<?xml version='1.0'?>
<rdf:RDF
   ...(namespace definitions)...
   <owl:Ontology rdf:about="http://.../example_ont">
     <rdfs:comment>An OWL ontology</rdfs:comment>
     <owl:versionInfo>e.g. the ontology's version and author</owl:versionInfo>
     <owl:imports rdf:resource="http://.../some_other_ont.owl"/>
     ...(additional import statements)...
   </owl:Ontology>
>
...(ontology concept declarations: classes, properties, individuals, annotations)...
</rdf:RDF>
```

Example 4: Defining administration information about an OWL ontology in RDF/XML

The OWL language constructs already include the RDFS vocabulary with respect to “class”, “subClassOf”, “property”, “subPropertyOf”, “domain”, “range” and the notation for declaring individuals of a class. Therefore I will constraint myself to a brief overview of what is new, respectively differs from RDF schema.

Classes In contrast to RDFS there are two implicit common classes in OWL. While “Thing” denotes the class of all OWL instances and the super class of all OWL classes, “Nothing” represents the class of no instances, and is the common sub class of all classes. A class in OWL Lite can be declared to be the *same as* another class (e.g. “railway” is the same as “railroad”) or also an intersection of classes. Additionally various *property restrictions* on the individuals can be applied when describing classes.

For example, the class of objects with engines could be given as the set of instances, that are *all* restricted to have the class “engine” as the range of the property “poweredBy”. Alternatively the restriction can also be based on only *some* values or a specific *cardinality*. However OWL Lite only support “0” and “1” for the cardinalities. In any case OWL Lite requires the restriction's range to be a class.

Properties A property in OWL Lite is either an *object property* which relates individuals to individuals, or *data type property*, which assigns a data value to an individual. In both cases the values for the properties domain and range must be classes in OWL Lite.

An object property can be additionally declared to be *symmetric* (e.g. “adjacent_to”), *transitive* (e.g. “located_in”) or *inverse* (e.g. “owns” is inverse of “is_owned_by”) to another one. According to the “*same as*” relation of classes, also properties allow for denoting their similarity.

Furthermore it is possible to restrict the range of a property to exactly one instance instead of multiple individuals of a class. This is done by declaring the property to be *functional*. Similar to this using the *inverse-functional* property restricts an object property's domain to one instance.

For example the functional property “has_driver” between the class “vehicle” and “person” denotes that there is only one unique person, who is the driver of a vehicle at the same time.

Individuals In addition to the declaration of individuals to be instances of a class, a pair of individuals can be stated to be *equivalent* to or *different* from each other. For example, “henry's car” could be the same vehicle, which is also identified by a certain car registration number.

Datatypes XML Schema data types can be used for assigning individuals certain values. For instance a person can be assigned a name and an age represented as a string respectively integer.

4.1.2.3. OWL DL and OWL Full

The OWL DL and OWL Full sub-languages share the same vocabulary, which extends the OWL Lite language constructs. Whereas OWL Full does not constraint the semantics of RDF language constructs, among other things OWL DL requires the disjointness of object and data type properties as well as the prerequisite, that an ontology class is not simultaneously an instance or a property.

Additional Vocabulary Classes can be declared to be *disjoint* with or to be the *union* of others. With respect to the previously explained property restrictions for defining classes, a class can now also be constrained to have only instances, which are equivalent to the domain of a property, that has a specified range instance or data type.

For example the class of FORD automobiles can be given as the class of all instances, that have the “has_manufacturer” property with the range “FORD”. A class can also represent the complement of another class, or can be defined by an enumeration of its possible instances, which is not allowed in OWL Lite. The language construct of enumeration may now also be applied to data types.

For example the range of the data type property “speed_limit” can be stated as one km/h-value of {30, 50, 80, 100}.

This concludes the introduction of the expressiveness of the OWL language family. Additional information and examples of the language constructs are given in [KOS03].

4.1.3. Discussion

OWL represents a well-designed language family that offers different levels of expressiveness, and provides a comprehensive set of language constructs with a strictly defined semantics (OWL Lite and DL) for modeling ontologies. In the case, that an ontology needs the se-

mantical freedom of RDF constructs, OWL Full can be used for this purpose. This is e.g. required to denote real class equivalence in addition to the similarity. That is not possible in the other two sub-languages, because here classes can not be individuals at the same time, which is necessary in order to declare their genuine equivalence.

OWL is already accepted in the scope of the Semantic Web, and can also be employed in ontology-based spatial context management systems. Its formalized syntax and semantics enables the sharing and reuse of information by new applications and other independent OWL-based systems. The OWL sub-languages are also supported by an increasing number of tools, that include several reasoners, which are capable of handling OWL Lite, DL and also restricted subsets of OWL Full. Above all the automated deduction of ontological knowledge is required by aware applications.

4.2. Reasoners

In the previous sections ontology languages, their expressiveness and reasoning support were discussed. This chapter gives a short introduction to selected ontology reasoners, that are suited for the task of deducing implicit spatial context from explicit location-based information.

4.2.1. Pellet

Pellet[1] has been developed at the institute for advanced computer studies at the University of Maryland. It is an open-source lightweight OWL DL reasoner, that is completely written in Java, and also provides its interface in this language. For querying RDF data Pellet provides an implementation of RDQL[2].

Besides its reasoning capabilities the supported tasks are checking of ontological inconsistencies, validation of class satisfiability and individuals, and reasoning about XML Schema data types [PS04]. A special feature is the automatic transformation of OWL Full RDF/XML ontologies to be valid OWL DL.

4.2.2. Jena2

This Java-based framework is the second version of the Jena reasoner. It follows from the HP Labs Semantic Web Research Program[3], and is now available under a BSD open source software license.

Jena2[4] supports reasoning with respect to RDFS and subsets of DAML and OWL. The framework provides several Java APIs for accessing RDF models and ontologies. Besides

1 The Web site for Pellet: "http://www.mindswap.org/2003/pellet/"
2 The **R**DF **D**ata **Q**uery **L**anguage is submitted by the W3C, see "http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/"
3 http://www.hpl.hp.com/semweb/
4 The Web Site of the Jena2 toolkit can be found at "http://jena.sourceforge.net/"

an implementation of RDQL Jena2 also allows for the persistence of ontology models in relational databases.

4.2.3. Racer

Racer is a complex and efficient description logic reasoner that has been designed at the Technische Universität Hamburg-Harburg[1]. Racer stands for “Renamed ABox and Concept Expression Reasoner”[2], and is the successor of Race (Reasoner for ABoxes and Concept Expressions).

In addition to other DL-related tasks (e.g. UML verification) Racer can handle RDFS, DAML+OIL, OWL Lite and OWL DL [HM04]. It also allows for persistence management for the assertions on concepts and those on individuals. The reasoner itself is implemented as a server program in Common Lisp for Linux and Windows. A Java interface is available for clients, that can connect to Racer via a socket connection over TCP/IP.

The latest version 1.8 of this reasoner (RacerPro) is commercial and can be be purchased from the company Racer Systems[3].

1 Documentation and download links can be found at “http://www.sts.tu-harburg.de/~r.f.moeller/racer/”
2 The term “ABox” is frequently used with DL system. It represents the assertional knowledge about individuals.
3 http://www.racer-systems.com/index.phtml

5. SCORE's System Design

This chapter presents the design of the spatial context management framework that is developed together with a first application in the scope of this thesis. It is called "*Spatial Context Ontology Reasoning Environment*" or simply "*SCORE*" in abbreviation. The framework's name corresponds to its design of combining efficient context models (see chapter 2.4.1) for the representation of low-level spatial context and ontology-based context representation and reasoning (see chapter 2.4.2) in order to adopt the advantages of both individual approaches as stated in chapter 2.4.3.

I would like to begin with the design goals, the framework has to address, and continue with the reasons why DWARF has been chosen to demonstrate the capabilities of SCORE. Next the decomposition of the framework's responsibilities is presented by means of an architectural overview, that is complemented by a detailed description of SCORE's two separated subsystems, one for the aggregation and federation of spatial context, and the other for the reasoning about it. After this more extensive chapter about the subsystems' designs, the persistence requirements and the design topics regarding the security of sensitive data as well as the user's privacy are explained. In the succeeding chapter SCORE's hardware requirements and the used off-the-shelf component are presented, followed by a discussion of the necessary considerations with respect to software control and the system's conditions on startup, shutdown and common exceptional cases, which concludes the design.

5.1. Design Goals

Before starting with the actual design of SCORE, several design goals have to be considered and discussed. These address to the quality attributes of a spatial context management system and its architecture as explained in chapter 2.3.4, and are categorized into qualities for performance, dependability, mobility and maintenance followed by the usability criterion. Several compromises between contrary criteria are pointed out at the end of this section.

5.1.1. Performance qualities

Above all SCORE should be designed to support Augmented Reality applications. As described in chapter 1.2 one of the AR characteristics is the interaction with objects in the physical world in real-time. As a matter of fact the diverse tasks for the management of spatial context information require a high performance.

Response Time After an application issues a request for low-level or interpreted spatial context, the system must process the query quickly enough to meet the real-time criteria of AR.

Throughput The time, which is measured between the moment of acquisition of physical or virtual spatial context information and the moment this information is available to the other framework components, has to be as low as possible in order to guarantee a high throughput of low-level spatial context. In addition the processing of context for aggregation and deducing higher-level information must be efficient so that a high frequency of context interpretations can be achieved.

Low Latency Time The amount of time required for the actual communication between all of SCORE's components and between the applications and the framework has to be very low.

5.1.2. Dependability qualities

The qualities according to *dependability* focus on the stability of a software system. Paying attention to all of the following attributes results in a system, which will mostly fulfill its tasks correctly and operate faultlessly no matter what happens.

Reliability All off SCORE's components must perform their tasks absolutely in accordance with their specification, since a deviating behavior could affect the dependability of the whole system.

Availability If requested, SCORE must run 24 hours seven days a week. It must always be available to continuously acquire and process spatial context, and provide its functionality to querying applications.

Robustness and Fault Tolerance In general sensor devices do not always provide correct and accurate data. In addition contextual inconsistencies and ambiguities have to be expected. Among other things also application requests may be specified invalidly. The system has to deal correctly with these possible sources of errors, and remain operational at all costs.

Security and Privacy Spatial context of users may represent confidential information. Thus the system has to effectively control the access to its data using an appropriate security mechanism, and guarantee the users' privacy, also in the case of system attacks and intrusion attempts.

5.1.3. Mobility qualities

These qualities apply to SCORE's scalability, the support for mobile devices and interoperability between different soft- and hardware components, that are used in the scope of the system.

Scalability SCORE must dynamically adopt itself to a dynamically changing number of possibly distributed spatial context sources and applications. It must also be able to handle changing numbers of physical and virtual entities.

Low Hardware Requirements AR applications are often deployed on mobile and wearable computers which do not provide the resources regarding computing power and available memory as stationary servers have at theirs disposal. They are also frequently running together with other systems on the same computer or also in environments, which do not provide external computing resources, to which parts of the context management tasks could be shifted. These issues result in the requirement that SCORE should be dependent on low computing resources.

Interoperability SCORE should be able to deal with the heterogeneity of external components such as the context providers, persistence management systems or applications, that are implemented in different programming languages and possibly run on different hardware and software platforms.

5.1.4. Maintenance qualities

The maintenance qualities consider the simplicity of any possible modification to the system once it has been deployed.

Modifiability, Extensibility and Evolvability All of SCORE's subsystems, but especially the reasoning component of the framework must be *modifiable* in order to change existing functionality or *extend* it according to the requirements of novel sensor systems and applications. This also includes the ability for a possible later *extension* regarding the management of general context in addition to spatial context information.

SCORE must also provide a specification, which allows the traceability of its *evolution*.

Adaptability Though SCORE is regarded to support applications in AR, it should also be able to address general spatial context-aware applications.

Portability The framework should mainly not be dependent on specific hardware or a particular software platform such as an operating system or a persistence management system.

5.1.5. Usability criterion

This is probably the most important quality attribute, since a system does not need to be built at all, if it is not usable and accepted by the persons who should work with it. Therefore SCORE should be designed a way, so that it facilitates its employment as much as possible.

Certainly this does not refer to the end user, who will only experience the AR applications built on top of the framework. However it refers to the system architects, who should be able to easily configure the system's behavior to the spatial context-aware applications they want to provide to the end users.

5.1.6. Compromises

Certainly not all of the qualities stated above can be addressed completely in limited time. Therefore several trade-offs have to be taken into account in order to present an effective and efficient spatial context management framework.

Development Costs vs. Functionality Not all of the framework's functionality, which is addressed in the system design has yet been implemented. The missing functionality corresponds to the resolving of contextual inconsistencies and the persistence management. However the possibility of an easy extension of the system is provided to meet these functionalities in a future development.

Development Costs vs. Security and Privacy Though some related context management systems (see chapter 3) have been criticized for not concerning security and privacy issues, also in this thesis these concerns are just investigated but not yet implemented.

5.2. DWARF as the deployment platform

SCORE consists of components which are designed to be as much independent of the underlying communication framework as possible. It can also be integrated into an existing spatial context management framework, such as the Nexus platform (see chapter 3.1), which lacks the context reasoning capability.

For testing and demonstrating the features of SCORE, DWARF has been chosen. As explained in chapter 1.3.2 on page 19 its middleware provides the foundation for a distributed service-oriented architecture by dynamically connecting needs and abilities of services. It also takes over the task for *resource discovery* in the network that may span multiple distant network nodes, and provides the foundation for selecting discovered services based on their quality-of-service (QoS) attributes ([MAC05], chapter 5). Because DWARF is aimed at supporting the mobility qualities of its applications, it allows for scalability, mobile and wearable devices as well as interoperability between different hard- and software components.

Furthermore DWARF has been used in a large variety of Augmented Reality applications,

and will be the basis for even new ones, which should be enabled to shift the management of spatial context towards the generic components of SCORE.

Section 1.3.1 already introduced to the principles of DWARF services. Because the following section discuses the decomposition of SCORE into subsystems by means of an UML 2.0[1]-based notation, this style of modeling, which is commonly used by the DWARF research group at the Technische Universität München, is briefly explained as follows.

A DWARF service is modeled just similar to a class in UML with the stereotype “<<service>>” and a name representing its functionality. Each need of a service is denoted as a semicircle and each ability as a UML interface, where both needs and abilities are linked to their corresponding services via full lines. The need and ability type is denoted textually with a preceding colon near to the need respectively ability. In order to state the communication protocol, the connector type is also often declared surrounded by a stroke-dotted rectangle. The connectors are used to establish a communication between services. In general these connectors are also able to support the connection of more than two services. For instance, a service can announce a need, which is satisfied by abilities of multiple other services. For more information about connectors, protocols and the general setup of a communication between DWARF services, the reader is referred to [MAC01] and particularly to [MAC05] (chapter 5.3. and 6.3).

Example connectors are “ObjrefExporter” matching “ObjrefImporter” (synchronous communication via CORBA method calls), “PushSupplier” matching “PushConsumer” and “PullConsumer” (asynchronous communication by means of CORBA structured events) and “Shmem” matching itself (shared memory communication between services on the same machine).

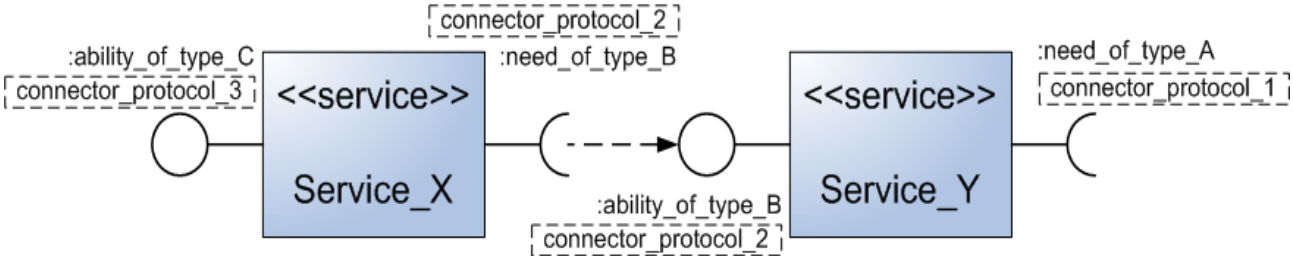


Figure 27: Two DWARF services with each one need and ability

Services can only connect to each other with the help of the DWARF middleware if their data types regarding the need of the one service and the ability of the other service and both connectors match to each other. Needs and abilities of different services, which satisfy this requirement, are linked via lines with an arrow indicating a dependency relation (stroke-dotted line) or data flow (full line).

The example in figure 27 depicts two DWARF services, which are allowed to be connected since the need of “Service_X” corresponds to the ability of “Service_Y”, and both services use a matching connector protocol.

1 The UML homepage can be found at http://www.uml.org on the Web.

5.3. SCORE's Subsystems

By means of the requirements explained in chapter 2.3 and the conceptual design of the combined approach regarding spatial context representation and modeling, which is presented in chapter 2.4.3, the both layered and service-oriented architecture of SCORE is decomposed into two major subsystems. These are the spatial context federation subsystem located on the federation layer and the spatial context reasoning subsystem on the reasoning layer. Before the structure of both subsystems is presented in detail, a conceptual overview of SCORE's relationship to its environment is given as follows.

5.3.1. Architectural Overview

Similar to the tiers of ontology [FRA03] (see table 2 on page 49) the general architecture is divided into different layers as depicted in the conceptual representation of figure 28 on page 102. Here SCORE is situated on the federation and reasoning layers. Please note, that there may be multiple instances of all of the services, which are shown in this figure. Therefore the following explanation of the individual layers refers to each of the depicted services in plural form.

Physical Layer The *physical layer* on the bottom accords to the ontology tier 0, and represents the physical reality, which contains real context entities such as users, animals or vehicles for instance.

Observation Layer The observation layer corresponds to the ontology tier 1. The sensor services on this layer acquire information about low-level spatial context of the entities in the real world, and provide this information to the services in the layer above. The connectors used for information exchange between the sensor services and the local AR model services is not important for SCORE's design, and is therefore not stated in figure 28.

Spatial Context Model Layer This is the layer representing the world, in which each particular object can be referenced by its individual name (ontology tiers 2, 3). The two tiers above this layer, which correspond to SCORE, are dependent on the separation of individuals by their names in order to guarantee the efficient management of spatial context of these objects. Though the ontology tier 3, which describes the social reality with respect to entities, does not intend for the anonymity of these, it is possible to emulate this approach by logically declaring anonymous entities by means of an enumeration, for instance.

In contrast to the ontology tier 2 objects are not restricted to represent only physical ones, but can also be virtually generated. The *spatial context model layer* contains model services that combine the observation layer's location-based information about real entities with virtual objects into local context models. They provide access to these basic data of spatial context ("SpatextBaseData", i.e. low-level location-based information) to one or more federation services on the layer above.

Federation Layer The *federation layer* also corresponds to the ontology tier 3, where an object can be found by its name in the environment. The responsibility of the spatial context federation subsystem is to mediate between applications or the reasoning services and the local spatial context models. The federation services aggregate "SpatextBaseData" of individual physical and virtual entities, which is retrieved from possibly multiple local AR models. By accessing a persistence management system on this layer a persistent history of low-level spatial context can be achieved in addition to a volatile history that is managed by each federation service.

Each spatial context federation service (SFS) also provides an interface ("SpatextBaseDataQuery") to applications which are dependent on instant responses for low-level spatial data queries regarding one or more managed entities. The same interface can be used by the spatial context reasoning services (SRS) to acquire the basic context information for their inference tasks.

Reasoning Layer This layer accords to the ontology tier 4, where new higher-level knowledge about entities and their spatial relationships is deduced from existing lower-level context. If an application uses the "SpatextQuery" interface to request interpreted information about physical and/or virtual entities, the spatial context reasoning service queries a SFS for the corresponding "SpatextBaseData", performs the reasoning tasks and replies the inference result back to the application.

Also on the *reasoning layer* a persistence management system allows storing and retrieving context information. However it does not manage low-level context, but only inferred knowledge, which can be reused in additional spatial context interpretations.

Application Layer Generally the *application layer* also corresponds to the ontology tier 4, since certain applications may operate as cognitive agents to deduce knowledge similar to the SRS. These applications may bypass the reasoning layer and directly query one or more federation services for not interpreted basic spatial context of entities.

Certainly applications can also access the local AR models to manage the virtual object representation in dependence of information about real ones. In addition an application can itself be again a context provider for SCORE. However this is not depicted in figure 28 in order to keep the diagram clearly arranged.

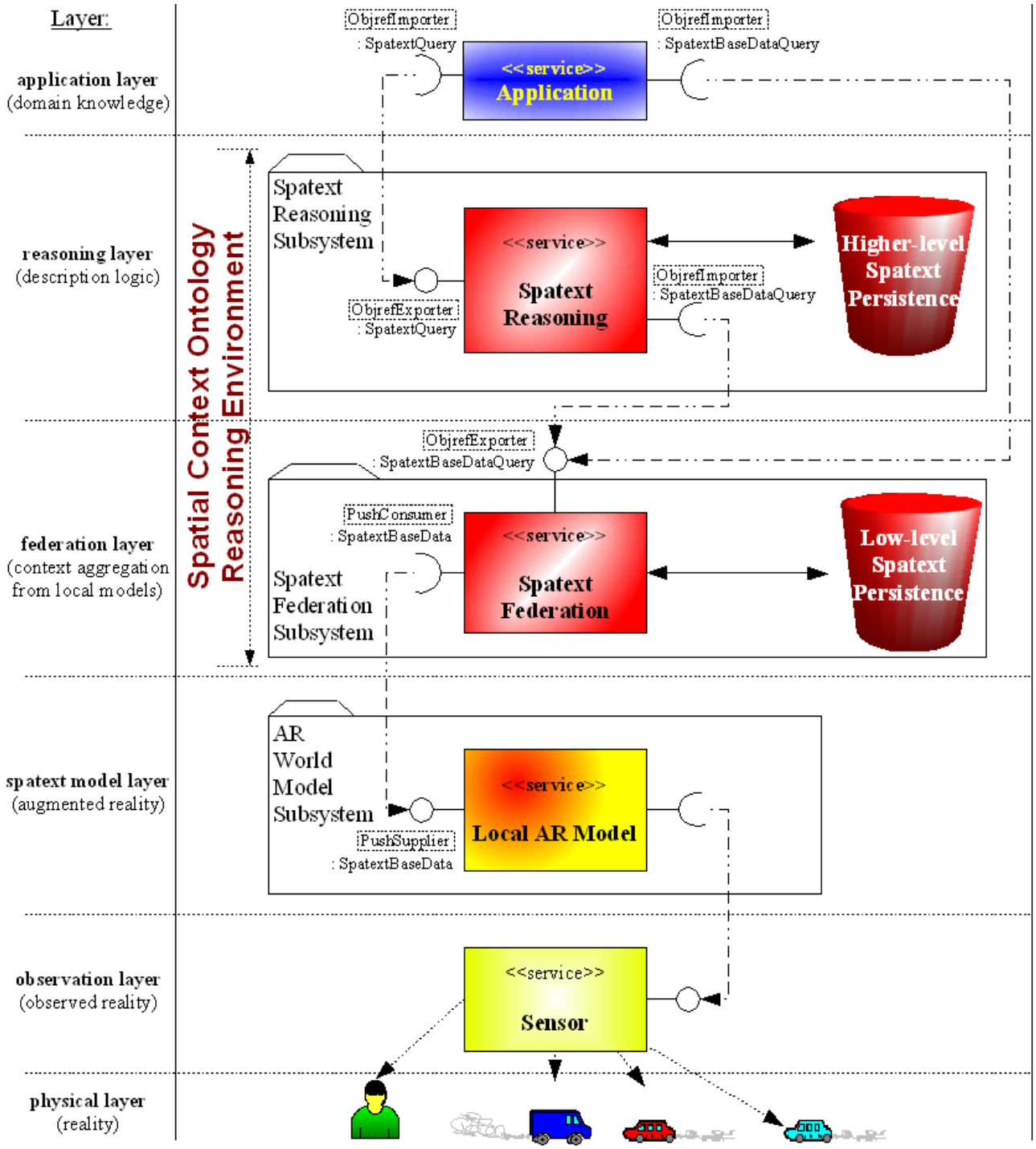


Figure 28: Schematic view on SCORE's architectural design ("spatext" denotes spatial context)

Design Rationale Above all the decision for a layered service-oriented architecture is founded on the scalability, distribution and modularization aspects. The layered architectural style provides logical data independence between components (services) on different layers. This enables the modification of components on one layer without any or with only minor effects to the constituents of the other layers. The service-oriented approach, which is addition-

ally pursued in SCORE's architecture, enables the loosely coupling of distributed services and their interaction so that changes to a service will not impair other services, particularly if the other services are situated on a different layer.

The presented overview of the architecture already involved the most important design principles for complex software systems in order to provide a maximum of flexibility. These are abstraction and information hiding. SCORE is divided into two major subsystems for spatial context federation and reasoning, where each of them focuses exactly on the assigned tasks (abstraction). Also the individual services SFS and SRS are able to conceal both their implementation logic – that means their algorithms – and their particular data representations (hiding). The only visible information to the applications are the data types used for communication between the services. The data types represent the communicated low-level context especially on the federation layer and the higher-level interpreted spatial context on the reasoning layer. This distinction between types of context and the free selection of the according query interface allows application developers to decide on their own whether or not the reasoning capabilities of SCORE are desired to be used. Certain AR applications may not be dependent on interpreted context, yet they possibly want to make use of a federation service in order to profit from its mediation feature and spatial context storage access.

As mentioned at the begin of this chapter the multiplicity of the federation and reasoning services may exceed one instance. This is due to the requirement of distribution and scalability. Having only one reasoning or one federation service would create a bottleneck and single point of failure like the approach of CoBrA[1] brings about. In order to enable context providers and applications to dynamically find SCORE's services, and in order to allow the reasoning services to find the federation services, a resource discovery mechanism must already be available in the distributed architecture. In case of DWARF, service managers are responsible for connecting matching services to each other. The requirements for a valid connection between services have been introduced in chapter 5.2.

As one may notice, SCORE does not include generic services for context acquisition, because a generalization for sensor services and the separation from application logic is hard to achieve as could be seen when discussing the Context Toolkit (see page 79). However if a sensor service or another spatial context source provides low-level spatial context, then the federation services are able to handle this information directly with or without the roundabout way via the local AR models. These are also related to the application domain to incorporate virtual objects and additional information in dependence of real entities.

Another concern is the distinction between the two different kinds of persistence management on the federation and reasoning layer. The reason for this design decision is the difference between the structure and complexity of the data to be stored. The federation layer manages low-level spatial context such as position and orientation information, speed and velocity data that can be efficiently handled by database systems with extensions for managing spatial data (e.g. IBM DB2 Spatial Extender or Oracle Spatial and Oracle Locator). In contrast the reasoning layer deals with higher-level context such as “a car overtakes a truck at a rounded hilltop” which can certainly be stored more flexibly in an ontology for instance.

1 The Context Broker Architecture is presented in chapter 3.2.

5.3.2. Spatial Context Federation Service

The *federation service* aggregates low-level spatial context information about physical and virtual objects from local context models, and mediates between these models and any services such as applications or reasoning services that require access to the distributed spatial context sources. In order to achieve a logical data separation of spatial context between different application domains, it is desirable to distinguish between groups of federation services for each application, where all individual groups are operating independently for their respective domains. However a federation service can also handle more than only one application simultaneously.

As seen in a first overview of the federation service (figure 29), it is equipped with one need and one ability. Due to the design goal of an easy usage of the service, the number of interfaces to its environment is designed to be small. Both the need and the ability are explained and discussed later in this chapter, followed by an introduction of “SpatextBaseData” as the spatial context data type with respect to the communication and aggregation of basic location-based information.

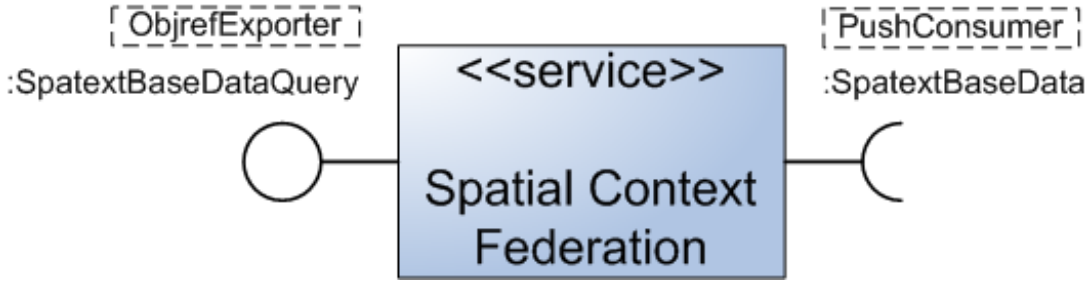


Figure 29: UML diagram of the Spatial Context Federation Service with one need and one ability

Rationale The aggregation of spatial context from various applications, that are not dependent on each other, results in higher costs for context indexing and context queries, because the context management for an individual application involves the strict separation of spatial information that belongs to the other applications. Nevertheless a federation service is able to work for more than one application in the case, these are cooperating in the same domain.

The restriction on only one need and one ability increases the usability of the federation service, because each additional interface would unnecessarily increase the learning curve for using it with applications.

5.3.2.1. The Service's Need

As depicted in figure 29 the federation service announces a need for event notifications (“PushConsumer”) regarding low-level location-based information about entities (“SpatextBaseData”). This information can be provided simultaneously by multiple spatial context provider services, which offer an ability of the same data type (“SpatextBaseData”) and use

the corresponding connector protocol for event-based communication, i.e. “PushSupplier”.

Design Rationale The decision for using an event-driven communication with the federation service's need interface is founded upon the asynchronous property of this communication mode. In contrast to synchronous method calls context providers are able to continuously send structured events of spatial information without having to wait for any reply. The federation service is dependent on exactly this continuous information delivery, because it must be able to instantly provide up-to-date basic spatial context to the reasoning subsystem and any querying application.

The asynchronous communication mechanism significantly increases the performance of the context providers, especially when the frequency of context acquisition is rather high. Though there is no 100 percent guarantee for the delivery of an event (events might get lost) the benefits in the flexibility of asynchronous communication clearly surpasses this limitation.

5.3.2.2. The Service's Ability

The federation service has a common interface for querying basic spatial context information about contextual entities via calls of the methods in the “SpatextBaseDataQuery” interface (see figure 29). In this case the service is capable of handling multiple queries at the same time. In general the methods provided by the query interface have a common parameter that tells about how many contextual history items should be returned per entity starting with the most current one. A *history item* refers to the low-level spatial context of an entity, that is measured at a discrete moment in time, and can also contain the current context. Each history item is encapsulated in an instance of the data type “SpatextBaseData” that will be presented in the succeeding chapter.

getPrimaryUserHistory retrieves a specified number of history items with respect to the spatial context of the primary user[1] if one exists.

getNearestEntityHistory returns a specified number of contextual history items of the entity, which is closest to the primary user if it exists.

getEntityHistories enables to query a specified number of contextual history items of one or more entities by optionally stating the individual's identifier and class membership. Dependent on its provided information “*getEntityHistories*” behaves differently as shown in figure 30.

1 The concept of the primary user is explained in the succeeding section.

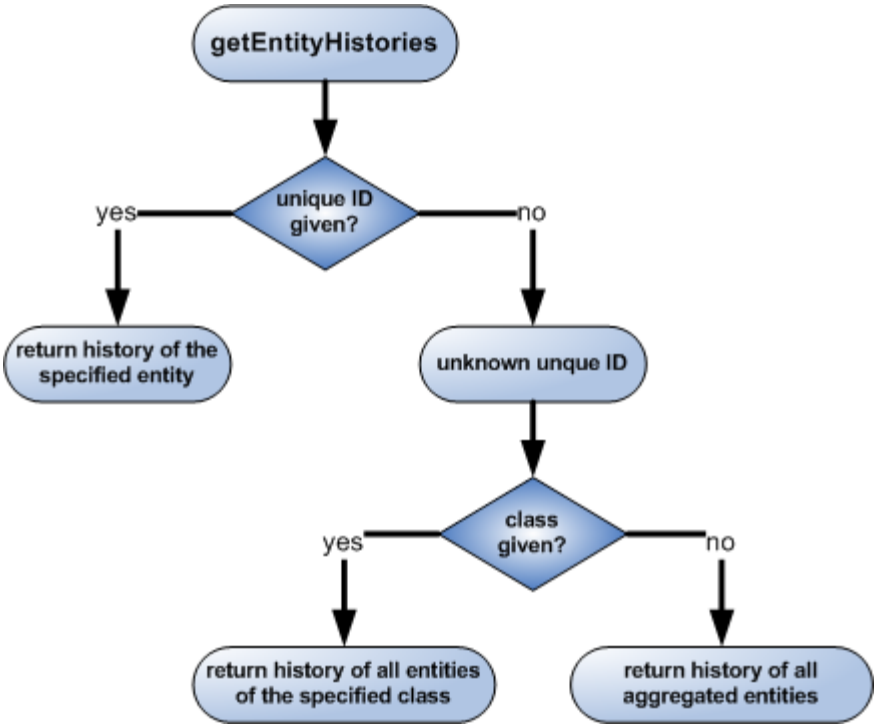


Figure 30: Flowchart diagram regarding the behavior of "getEntityHistories"

If the entity is identified by its unique name, then the history, which belongs to this entity, is returned. By additionally specifying the class, this entity belongs to (e.g. "person" or "automobile"), the query processing efficiency can be increased.

In the case, that the entity's name is not given, however its class is provided, the history of all entities belonging to this class is returned. If also the class is not stated, then the specified number of history items of all aggregated entities is retrieved.

getNearbyEntityHistories returns the given number of contextual history items of all entities that are in a specified maximum range of the primary user if it exists.

For convenience, the query interface of the federation service additionally provides the methods "getPrimaryUser", "getNearestEntity", "getEntities" and "getNearbyEntities", which represent special cases of the ones above. These methods implicitly set the number of context items to "1", so that the most up-to-date spatial context information is returned.

Application developers who want to make use of SCORE's federation subsystem are referred to chapter 10.1.1 in the appendix, where the federation service's interface description is listed.

Design Rationale By using a method call-based interface I decided for a synchronous communication mode, which blocks the calling system while the query is processed. However, in

contrast to the need interface, where the federation service only receives events, here the communication is based on a client-server approach, where the client such as a reasoning service specifies additional attributes for constraining the possible results of its query. Events are far too inflexible and inefficient for the transmission of both the dynamically declared queries and the varying responses. Another criterion is the amount of event channels for event-driven communication. Each event channel means an additional loss of computing resources, and since channels would be required on both sides of communication, two for the delivery of the query, and two for the response, this could impair the design goal for performance.

The explained interface methods provide sufficient flexibility for querying basic spatial context information. The additional specific methods are provided for convenience, and do not raise the complexity of the subsystem. They are designed in accordance to the *command pattern*, which decouples the interface methods from the actual control methods [BD00].

The involvement of a query method parameter, which describes the number of discrete spatial context history items for each entity, is generally required by any service that deduces higher-level context on the basis of historical information. For instance, if a context source does not provide the orientation of an entity, then it can be deduced by the reasoning service via comparing the entity's positions which change over time.

5.3.2.3. SpatextBaseData for Modeling Low-Level Spatial Context

SCORE introduces two new structured data types (“SpatextBaseData” and “SpatextData”) for the communication between the federation and reasoning subsystems and its service environment. Special efforts have been dedicated to keep the number of new data types and the complexity of their usage as minimal as possible in order to decrease the cost of time required for setting up the communication between SCORE and its applications. The both structured data types refer to low-level respectively higher-level spatial context, and are designed to be both most flexible and easy to handle.

The specification of these data types contributes to the logical data independence, which means, that applications should not have to be modified if the context representation of the federation or reasoning subsystems is changed. Since this has to be considered in the design of SCORE, this section and a corresponding section in the chapter about the reasoning service explain their structure, and give reasons for the ideas regarding their individual design.

The data type “SpatextBaseData”[1] is used for representing low-level location-based information in the federation subsystem. It is provided to the federation services by the various spatial context sources, and is also used with the communication between the federation service and the reasoning subsystem or an application, which only requires low-level spatial context.

Figure 31 depicts the general structure of the “SpatextBaseData” IDL[2]. It provides spatial information about exactly one physical or virtual entity in the social reality (ontology tier 3). That

1 The declaration of “SpatextBaseData” is listed in the appendix in section 10.2.1.
2 The **I**nterface **D**efinition **L**anguage defines the interfaces between CORBA-based applications.

means, that each entity can be referred to by its individual name, and belongs to a class of entities which share the same properties[1]. As a matter of fact an instance of “SpatextBaseData” is uniquely identified by the name of the context entity it belongs to, and states information about the entity's class membership. The initial naming of entities and the assignment to their corresponding classes is the task of the context provider services, which have explicit knowledge of the application domains, and in particular know the identity of the entities, for which they provide the spatial context to SCORE. It is important to provide unique identifiers for all entities referred by all applications, because the federation service can serve more than only one application at the same time, and therefore must be able to distinguish between those entities.

The primary context[2] of an entity is represented by the mentioned unique identifier and the entities position and orientation at a particular time, which is encapsulated in the reused “PoseData” IDL from DWARF. Besides this important spatial information “SpatextBaseData” contains an extensible set of spatial attributes such as velocity and acceleration.

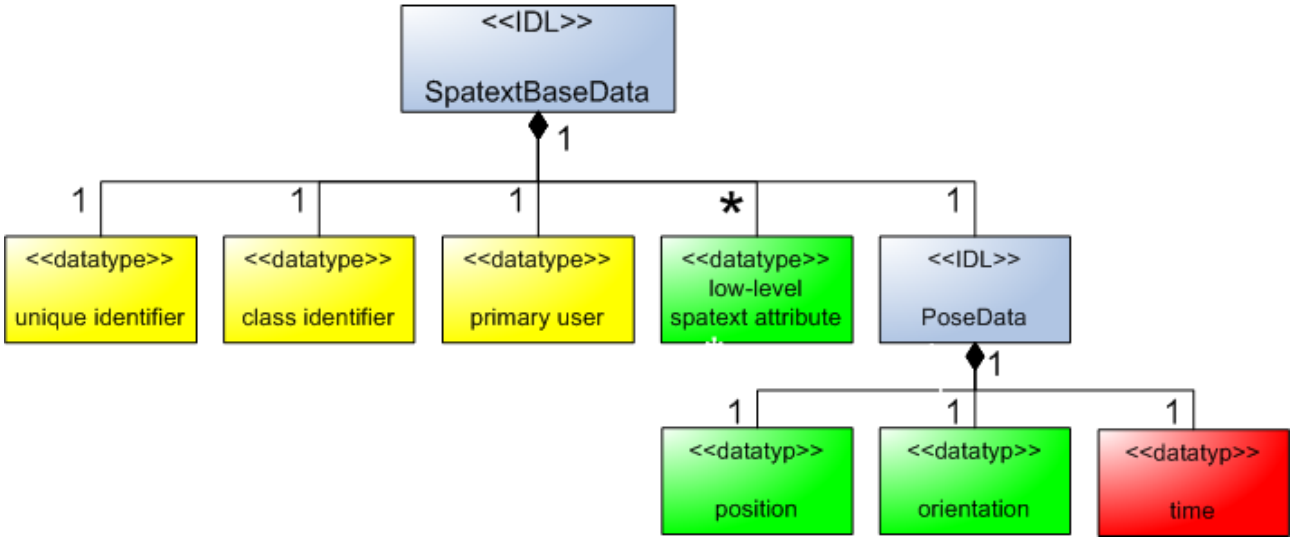


Figure 31: The constituents of "SpatextBaseData" in UML notation (less important data types of “PoseData” are hidden)

Another information, which is contained in “SpatextBaseData” and can be given optionally, is the statement, whether the described entity is the so-called “primary user”. That means, that there can be an entity of special interest in the application domain (e.g. the user of a context-aware AR system), whose spatial context should be treated preferential.

Since not all context sources are able to provide position, orientation and additional low-level location-based attributes, only the statement of the unique identifier, the class identifier and the according timestamp is mandatory (specifying the primary user attribute is also optional).

1 In the case you are a developer of an application or context provider service please also read chapter 5.3.3.4 starting on page 126. There the important requirements for naming the used classes are explained, because they have to follow the rules of a general terminology.

2 The difference between primary and secondary context is explained on page 34.

Design Rationale An additional modeling of low-level context for the representation within the federation service is not necessary, because the federation subsystem does not provide any means of context interpretation, but solely has to aggregate the “SpatextBaseData” items of the contextual entities. As seen before, this data type covers all important spatial context attributes, which are necessary for the representation of primary and secondary low-level location-based information.

Since the structure of this data type may nevertheless be changed in the future, the federation subsystem must be designed to be as much independent of it as possible. So additional spatial context attributes such as the lateral acceleration (which could also be deduced by the reasoning subsystem) can be added without any effects to the federation services. If you are interested in a more detailed description about the representation of explicit spatial context, and the implemented indexing mechanisms, you might want to also read section 6.2.2 in the chapter about the federation service's implementation, though this chapter is quite technical.

By introducing an unique name for each context entity in a particular application domain, this identifier can be used together with the time value as a lightweight primary key for efficiently indexing and searching the entity's low-level spatial context at a particular moment in time, or retrieving its accumulated contextual history.

Also the optional primary user property can be used to speed up query processing, since the necessary calculations for spatial queries regarding the location-based relations between the primary user and the surrounding entities can be preprocessed in advance whenever spatial context updates are delivered to the federation services.

The assignment of classes to entities is a fundamental design concept for SCORE. It will facilitate the reasoning process, because the entity class will be matched to an according ontological class as explained later in the chapter about the spatial context reasoning service.

5.3.2.4. The Service's Constituents

Now that the interfaces of the federation service have been presented the design of its internal components will be investigated (see figure 32). These consist of a central broker, an event handler for low-level spatial context events, which are sent by context provider services, a warehouse for aggregating a partial history of the received context in a main memory data structure, and a persistence provider that stores and retrieves persistent contextual history of entities. In order to explain the functionality of the broker the following sections first introduce to its surrounding components. After this the broker itself is presented followed by the discussion of the federation service's design.

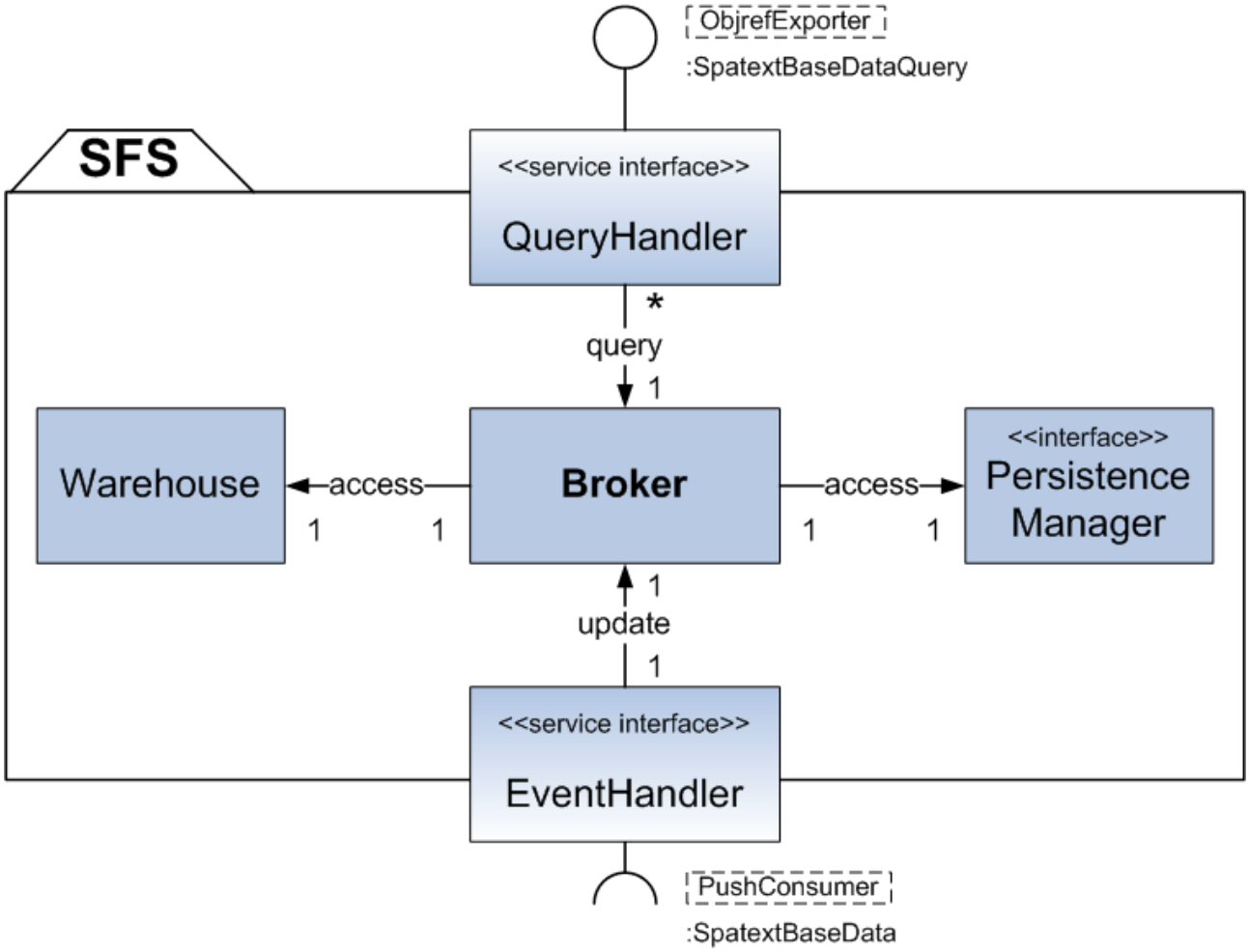


Figure 32: The structure of the Spatial Context Federation Service (SFS) using UML notation

5.3.2.4.1. Event Handler

The event handler is responsible for encapsulating the service's need (see section 5.3.2.1). Its task is to receive any “SpatextBaseData” events provided by any services, and forward each unpacked “SpatextBaseData” item to the broker.

5.3.2.4.2. Query Handler

The query handler is based on the ability service interface, and provides the implementation for the previously mentioned service access methods (see section 5.3.2.2). Per each querying service there is one query handler, which forwards the service's request to the broker, and returns back the broker's query result.

5.3.2.4.3. Warehouse

The warehouse represents the encapsulation of an efficient data structure for storing a configurable number of history items of entities' basic spatial context information. Since the broker hides the warehouse's internal structure from the other components, this design allows for exchanging its implementation for another one if this would be required in a future project.

The tasks of the warehouse are to manage the caching of explicit spatial context information ("SpatextBaseData") in main memory to increase the overall query efficiency, and to calculate the distance between the primary user and the other entities for increasing the processing speed of range-based queries.

5.3.2.4.4. Persistence Manager

The persistence manager defines the necessary logic for accessing the interfaces for external persistence management systems such as file-based storage, XML databases, relational or object-oriented databases for instance. Depending on the frequency of arriving spatial context, the persistence manager does not store each particular data item, but dynamically adapts the frequency of storage operations to such an extend that the individual persistence provider is not overloaded. The persistence manager is also responsible for retrieving previously stored information as historical context on behalf of the broker.

5.3.2.4.5. Broker

The broker plays a central role in the federation service's structure. It mediates between the handlers and the spatial context repositories. Though there is only one broker per each federation service, it can handle multiple requests at the same time. On the arrival of new event data for an entity, the broker accesses the warehouse and the persistence manager to store the spatial context item in main memory respectively on a persistent medium (figure 33).

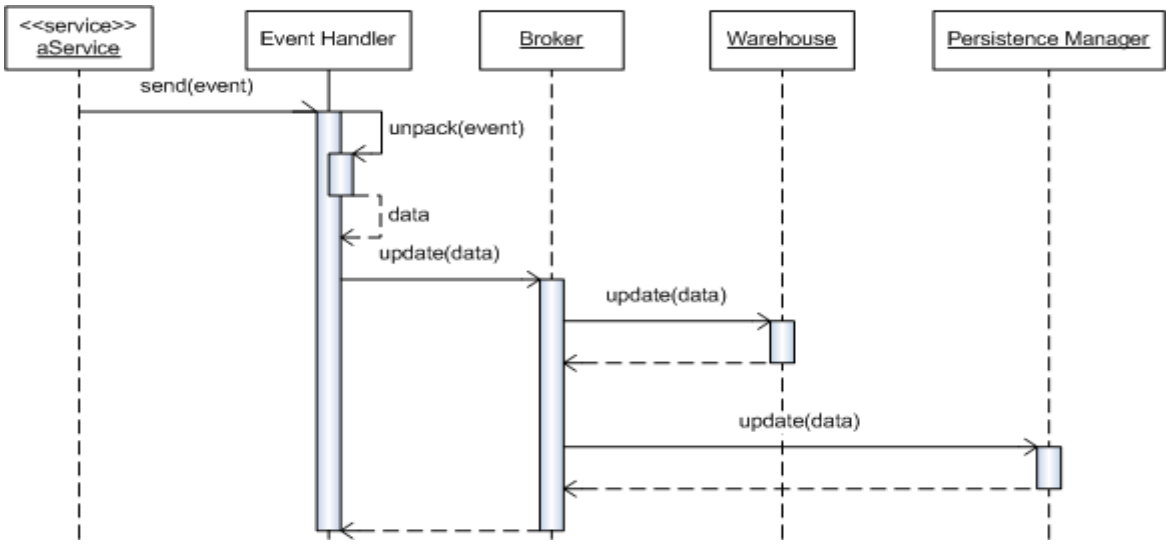


Figure 33: SFS workflow on the arrival of a new "SpatextBaseData" event (UML sequence diagram)

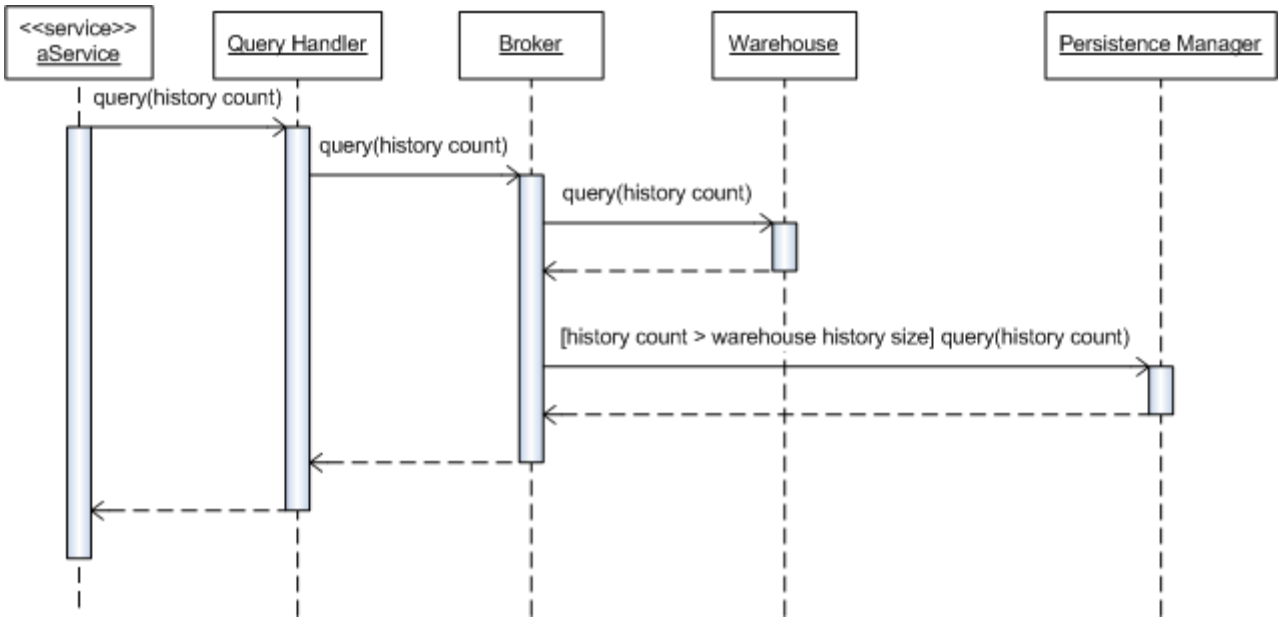


Figure 34: SFS workflow in case of a query with a certain number of contextual history items (UML sequence diagram)

If the broker is accessed by one of the query handlers requesting spatial context about entities on behalf of the reasoning services or applications it first queries the warehouse about the history of the addressed entities. As explained when describing the service's ability in the previous chapter, a query also has to specify the number of history items to return for each entity, i.e. the *history count*. In the case, that the main memory-based data repository of the warehouse does not contain enough information to match this number of required history items, the broker additionally retrieves the missing history from the persistence manager (see figure 34).

5.3.2.4.6. Design Rationale

The design of the federation service is founded on the facade pattern [BD00]. It enables to reduce the dependencies between classes by hiding the complexity of individual classes to the ones, which are logically associated with it. Therefore the event and query handlers are not required to know how to address the warehouse and the persistence manager, because the central broker mediates between them. Due to the low coherence this also allows for exchanging the other components of the federation service without serious effects on each other.

As one could see, the persistence manager is not designed to implement the file-oriented or even physical storage structures on its own. One reason for this is founded on the design goal of *portability*. Selecting a particular fixed persistence provider could imply, that numerous lines of programming code have to be changed, if SCORE was ported to another hardware platform or to another operating system. The other reason is based on the principal idea of

software engineering to reuse existing software modules or complete frameworks. Therefore the persistence manager's task is to mediate between the broker of the federation service and the actual persistence provider such as a database management system or a XML-based framework that can be used for achieving persistence for spatial context information.

The decision for a volatile data structure for caching spatial context in addition to the persistence mechanism has been made due to the fact, that the frequency of query processing of persistence providers is rather low compared with a main memory-based approach. This decision particularly shows to advantage when the frequency of arriving events and the frequency of queries is extremely high as required for Augmented Reality applications. In this field it is often not possible to omit the storage of intermediate events, since the missing information is mostly needed for the continuous reasoning about real-time effects.

5.3.3. Spatial Context Reasoning Service

On behalf of applications the *reasoning service* deduces higher-level spatial context from basic information. That means the reasoning service is located on the highest tier of the five tiers of ontology (see table 2 on page 49). In order to access low-level spatial information for the reasoning purpose, the necessary data can be queried from available federation services as a basis for the interpretation tasks. Similar to the federation service, it is also possible to have multiple instances of the reasoning service that can be distributed across the network in order to provide their inference capabilities to different applications. Nevertheless one instance of the reasoning service is also able to serve multiple applications at the same time.

Rationale Reducing the number of applications for each reasoning service decreases the services load and therefore also lowers its response times. If there are multiple instances of a reasoning service running at the same time, where each of them is able to access more than only one federation service, the performance of SCORE's subsystem functionalities is improved noticeably.

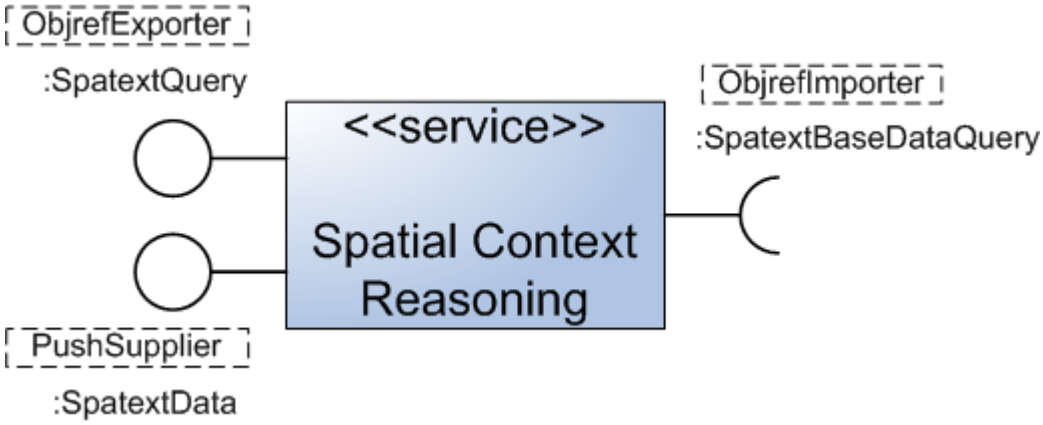


Figure 35: UML diagram of the Spatial Context Reasoning Service with one need and one ability

Figure 35 shows the external view of SCORE's reasoning service. It is equipped with one need and two abilities, which are presented in the following two sections.

5.3.3.1. The Service's Need

The reasoning service is dependent on the federation services' capability of processing queries regarding aggregated low-level spatial context of federated context sources. This dependency is based on the reasoning service's fundamental functionality of deducing higher-level information with respect to the spatial relationships of entities from their low-level context. In order to request basic spatial data about the entities of one or more applications, a reasoning service can call the query methods that are provided via the corresponding interface represented by the federation service's ability (see chapter 5.3.2.2). The query responses are supplied in the form of collections of “SpatextBaseData”, which serves as the basic type of spatial context representation for the communication between the two kinds of SCORE services[1].

Design Rationale The reasons for the decision of using synchronous method calls with the service's need instead of an event-based approach have already been stated in the chapter about the ability of the federation service (pages 105-107).

5.3.3.2. The Service's Abilities

Similar to the federation service also SCORE's reasoning subsystem enables other services to query for spatial information about entities. In addition the reasoning service also uses event notification to send contextual events to its subscribers. Therefore aware applications are able to actively request or be informed about higher-level context, which is processed on the basis of a terminological context representation and the dynamically changing location-based context of those entities, an application is interested in.

In its method interface the reasoning subsystem offers several methods. The first one is designed to respond the interpretation results that accord to any possible queries. The other methods can be used by applications for (un)subscribing to queries.

query returns low-level or higher-level spatial context of entities depending on a given query statement.

subscribe allows an application to subscribe for events regarding a specified query. The reasoning service notifies the application about the queried spatial context in a given frequency.

1 The declaration of both “SpatextBaseData” and the query interface are given in the appendix in chapters 10.2.1 and 10.1.1.

unsubscribe can be used by an application to cancel its subscription with respect to a certain query.

unsubscribeAll should be used if the application does not need any more notifications by a spatial context reasoning service.

Certainly it is not easy to restrict the complex context query functionality to only one parameter of an interface method. However due to the design of a query language, whose structure is even rather simple, and the involvement of pre-defined extensible terminological knowledge about application domains, this can be achieved.

5.3.3.2.1. Query Language

As known from chapter 5.3.2.3 about the low-level spatial information representation by means of “SpatextBaseData”, each entity with respect to all applications can be identified by its unique name, and belongs to a class of entities that share common properties. The query language benefits from this fact, because it allows for specifying entities by its individual name or also by its class.

property domain	`([:CLASS:] | \*)=([:INSTANCE:] | \*) (\b(as primary user)\b)?`
binary relation	`[:PROPERTY:]`
property range	`([:CLASS:] | \*)=([:INSTANCE:] | \*) (\b(as primary user)\b)?`

Table 6: The query language of the reasoning service in regular expression notation

The syntax of the query language is given in the right column of table 6, where the language syntax is written separately in three lines to increase their readability, though a concrete query is composed into one continuous character string. In this syntax a complete query statement is constructed of a binary relation (the property) between its domain and range of entities, where each of them belongs to a known or unknown class. Both the domain and range expressions are composed of an entity class and an instance statement (an individual entity) belonging to this class, where for each statement either a name can be provided, or simply the character “*****” can be used to denote all possible classes respectively entities. In the case, that an application knows whether the stated entity instance, which refers to the property's domain or range, is the primary user entity, it can optionally specify the “as primary user” expression to increase SCORE's processing performance.

The following examples demonstrate possible queries for higher-level spatial context of entities by means of different scenarios (e.g. traffic, office environment, etc.).

1.:	`Vehicle=car007 overtakes Car=car001 as primary user`
2.:	`Lorry=* isSpatialObstacleFor Motorcycle=bike27`
3.:	`Person=Henry as primary user locatedIn OfficeRoom=*`
4.:	`*=c3po collidesWith *=r2d2`
5.:	`*=* closeTo *=*`

Example 5: Using the query language of the reasoning subsystem

The queries in example 5 request various information about entities in various application domains. For instance, there is a query (1.) about whether the entity “car007” of class “Vehicle” overtakes the primary user entity “car001” of class “Car”. Another one (2.) wants to know all lorries, which are representing a spatial obstacle for the motorcycle “bike27”. Certainly there could also be an application, which needs to know in which office room the primary user “Henry” is located (3.) or whether the entity “c3po” collides with “r2d2” (4.) though their classes are not known. The last example (5.) represents a query for all entities of all classes that are close to all other entities of all classes.

In addition to the naming of entities by their unique identifiers the queries in example 5 also use simple and short names to reference classes and properties. These names correspond to an extensible global terminology representing all the classes and properties of importance to the applications. As explained later when discussing the context representation and modeling of SCORE's reasoning subsystem, this terminological representation also includes entity classes and their relations, where both of them are either addressed by short names, or can alternatively be referred to as resources on the Web (RDF, see page 85 for a description).

Using short class and property names is one possibility of querying higher-level context as one could see in the previous example. Those names can be used conveniently when there is agreement on the meaning of them by all the developers of applications, that use the reasoning subsystem of SCORE. In general however similar to the prerequisite of identifying individual entities by their names, there must be the capability of referencing their classes without the involvement of ambiguities. Such an ambiguity would arise if the class “car” is used as “automobile” in one application, and as “railway car” in the other. Certainly application developers should agree on a common terminology. However it should not be the responsibility of SCORE to force this.

Therefore SCORE also allows for the guaranteed unique identification of classes and properties by means of URIs, in spite of the fact, that a class or a property may be addresses from different points of view. In order to specify a query by providing unique uniform resource identifiers for its classes and its property, they must be declared using their fully qualified Web resource addresses. Of course in this case, the “*****” expression is not used for the classes and the property. The following table shows the first and the second query of example 5 with fully qualified names for the common class and property resources.

1.:	`http://.../TermsA#Vehicle=car007 http://.../TermsB#overtakes http://.../TermsC#Car=car001 as primary user`
2.:	`http://.../TermsX#Lorry=* http://.../TermsY#isSpatialObstacleFor http://.../TermsZ#Motorcycle=bike27`

Example 6: Using the query language of the reasoning subsystem with fully qualified class and property names

It is also possible to query for past information (e.g. “User=* wasLocatedIn Room=r1”). However since this explanation requires additional knowledge about the terminological representation of spatial context first the design of the reasoning subsystem has to be presented. The same issue refers to the introduction of the kind of knowledge representation for the responses of the queries (in form of collections of “SpatextData”), which is based on the user-defined reasoning feature of SCORE.

5.3.3.2.2. Design Rationale

The two abilities of the reasoning service allow for both synchronous method calls and asynchronous event-based communication. Therefore the applications developers are able to decide on their own whether they implement a mechanism for continuously querying the reasoner or subscribing to it in order to be informed about spatial context via events. The first approach should be used when the application involves multiple and often changing queries, which would imply a subscription for each new query and the cancellation of a subscription for each obsolete one. Events should be preferred if an application requires a rather small number of static queries. In the case that there is a large number of static queries, the usage of multiple reasoner instances should be considered, that access the same federation services.

Rationale of the Query Language The syntax used for specifying queries for spatial context is based on a global terminological representation of entity classes and their relationships. Since the classes also correspond exactly to those managed by the federation services, the query mechanism provides a sufficient capability for accessing both low-level and higher-level contextual information from the federation and from the reasoning subsystem, which additionally interprets the explicit context. The introduction of the “*****” expression allows for retrieving data about an known entity of an unknown class, unknown entities of a known class and all entities of all classes. This enables a high flexibility for stating queries.

The optional statement about the primary user for the domain or range of a property is most useful for the increased efficiency of the SCORE. The information retrieval regarding the primary user has to be most efficient since a lot of queries are referring to this entity (in fact it is possible in constant time as described in the implementation chapter 6.2.2.2 starting on page 149). Because the reasoning service frequently has to obtain low-level spatial context from the federation subsystem, it can pass the information, that an entity is the primary user, to the federation systems. Thus the response times for both subsystems are noticeably decreases. This fact is increasingly important, since the most queries are issued with respect to the primary user.

Using the uniform resource identifiers known from RDF allows for uniquely identifying the given classes and the stated relation between the classes' instances respectively the entities. Therefore it is possible to use the same class names in different applications without impairing each other. However, as could be seen in the examples before, the strict and sometimes awkward statement of the full qualifiers can be given optionally, and may be omitted in the case, that naming ambiguities will definitely not arise. This is possible if there is only one application or if the entity classes and relations in all applications are known to be distinct, or – in the best case – if application developers agree on a standardized terminology.

Because the query language is designed to be as simple as possible though it can be employed very flexibly, it contributed to the quality criterion of *usability*. Its syntax can be learned very fast, and it is even possible to understand its usage right away by looking at only one or two examples.

5.3.3.3. The Service's Constituents

The principal task of SCORE's reasoning service is the processing and deduction of spatial context on behalf of querying applications or applications that subscribe to certain queries. As its central component, the controller of the reasoning service mediates between the service's interfaces, the subscription component, the query parser, the reasoner subsystem and the persistence manager (see figure 36).

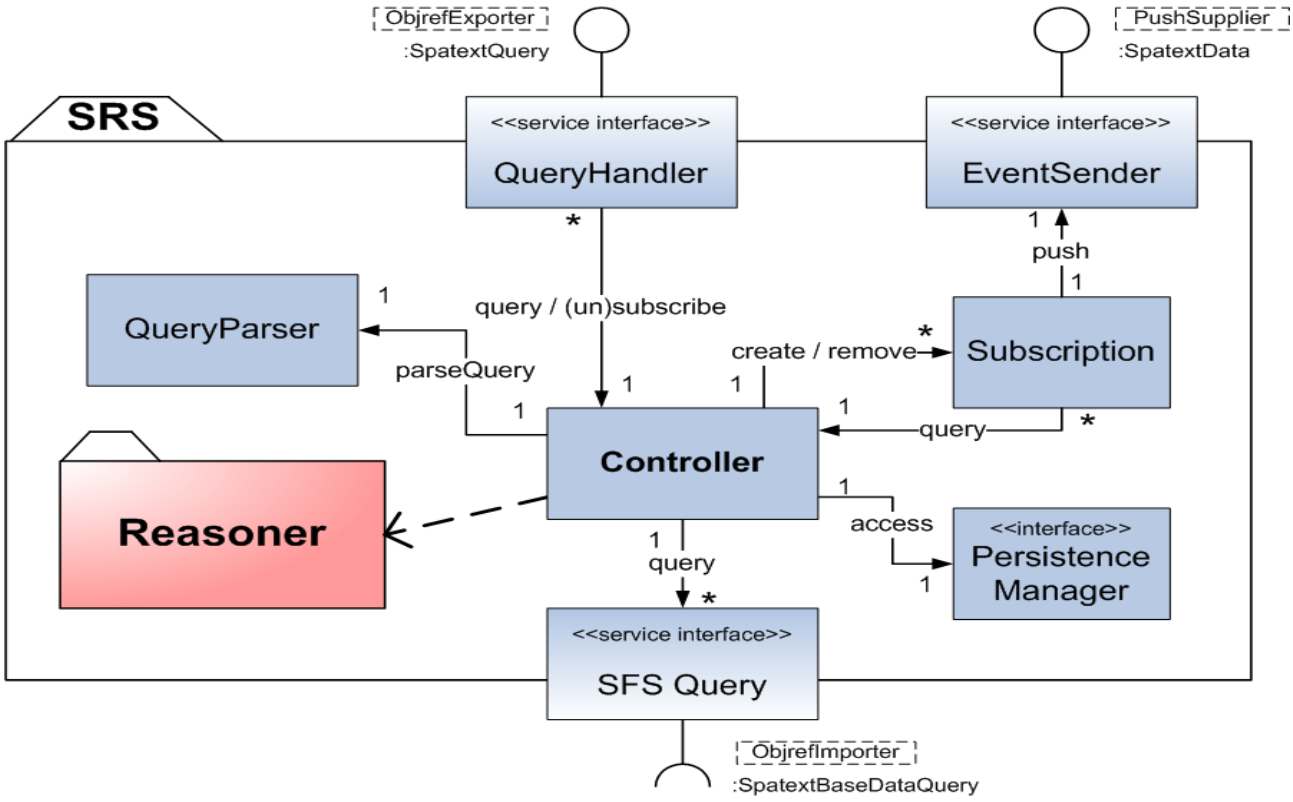


Figure 36: The structure of the Spatial Context Reasoning Service (SRS) using UML notation ("spatext" denotes spatial context)

5.3.3.3.1. Query Handler

The query handler is responsible for receiving spatial context queries, subscriptions for queries and requests for canceling one particular or all existing subscriptions of applications that are aware of spatial context in the users' environments. In order to provide their functionality, the query handlers access the service's controller, which manages the execution of queries and creates or removes subscriptions.

5.3.3.3.2. Subscription and Event Sender

Applications, that have to be informed about spatial context in the users' environments in certain intervals, can subscribe to context queries with the reasoning service. This can be achieved by a service description, where the required queries are explicitly registered for an application, or more flexibly by calling the subscription-related methods in the service's query interface.

Per each application, which requires event-based notifications about queried spatial context of entities, one subscription component is assigned. If an application request a new subscription for the first time, the controller creates a new instance of the subscription component, and provides the actual query and the notification frequency to it, where both have been specified by the application or are stated in the service's configuration. In the case, that an application requests an additional subscription though it already subscribed to at least one query, the controller searches for the corresponding subscription instance, which is responsible for this application, and adds the new query to it (see figure 37). In their given notification intervals each subscription component issues the queries to the controller, from which it obtains the requested implicit spatial context information. This data is then forwarded to the event sender, which packs it into a structured event and finally dispatches it in order to notify the subscribed application.

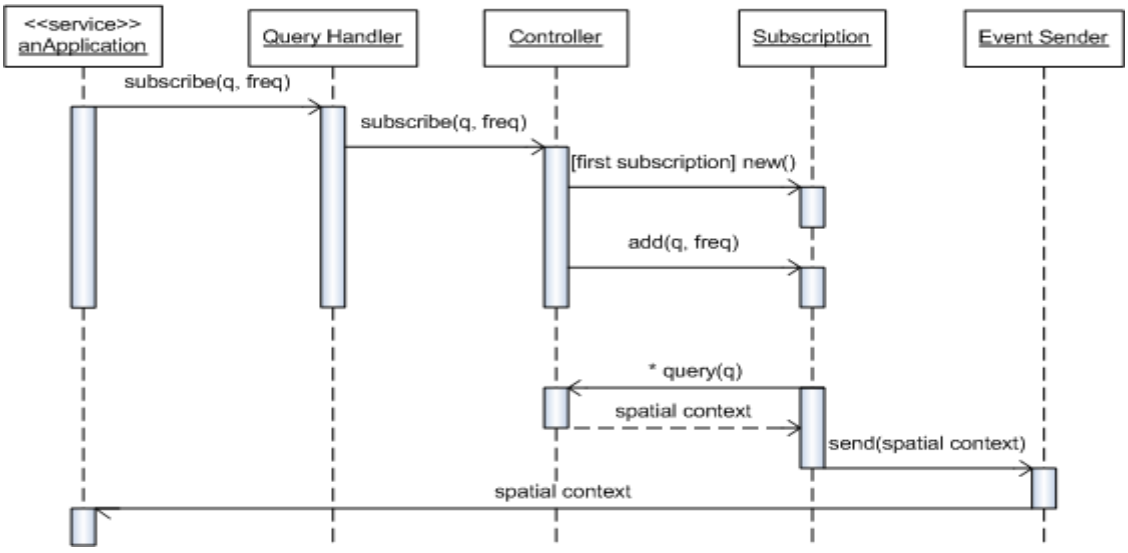


Figure 37: SRS workflow for subscriptions and event notifications (UML sequence diagram, "q" denotes query and "freq" frequency)

Similar to this procedure, the controller can also remove queries from the subscription component on behalf of an application. In this case the instance of this component is deleted, if it does not contain any more queries. This is shown in the diagram in figure 38.

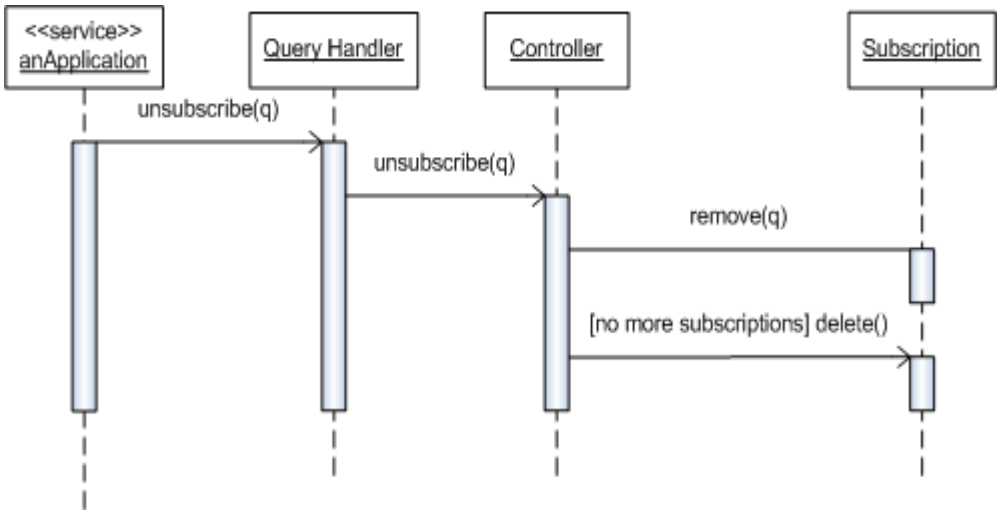


Figure 38: SRS workflow for canceling an existing subscription (UML sequence diagram, "q" denotes query)

5.3.3.3.3. Query Parser

This component is only invoked by the controller if it receives a new query or a subscription to a new query. The parser analyzes the given query, which is specified in the reasoning subsystem's query language (see chapter 5.3.3.2.1), and returns the parsed query back to the controller as a query item in a processable abstract syntax.

5.3.3.3.4. Reasoner Subsystem

As shown in figure 39 on the next page, the reasoner subsystem encapsulates the tasks for deducing spatial context (i.e. "spatext"). It provides the access to the description logic-based reasoner (DL Reasoner) and the user-defined context reasoner (Rule-Based Reasoner), which is composed of the rule parser and the rule invoker. The difference between the reasoners for static and dynamic spatial context is discussed in chapter 2.4.2.1 respectively 2.4.2.2.

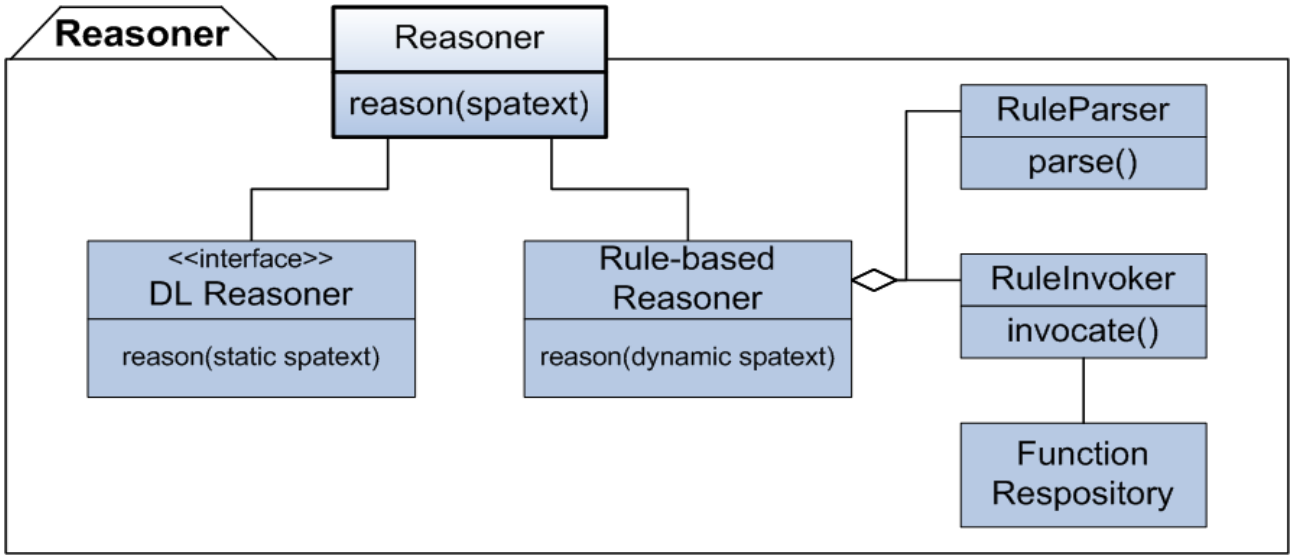


Figure 39: Reasoner subsystem of the spatial context reasoning service designed with the UML class diagram

DL Reasoner As denoted by the UML interface stereotype, a concrete DL-reasoner is not designed to be implemented in the scope of this thesis. In contrast an off-the-shelf ontology-based reasoning framework is selected, because ontologies provide a maximum of flexibility when declaring common terminological information about application domains. The selected reasoner which can be accessed via the DL reasoner interface will be presented in a later chapter about used third-party software. However it has to satisfy several common reasoning capabilities, since it is responsible for deducing general concept-based information of the entities' classes and their relationships in the application domains. Therefore among other things it must be able to check for class and property subsumption, equality and inequality, satisfiability and consistency of the terminology. In addition, based on the abstract information in the provided query, the reasoner must also be capable to resolve the stated class and property resources and to validate the class membership and applicability of the property to the entities which are specified in the applications' queries.

Finally the DL reasoner retrieves the rules corresponding to the individual query, and returns it to the controller for caching them. Those rules are later applied to the dynamic spatial context by the rule-based reasoner.

Rule-based Reasoner In contrast to the DL reasoner this one does not reason about static information, but enables the interpretation of dynamic spatial context of virtual or real entities referred to by the applications' queries, which are provided in an abstract syntax. In this case the necessary information about the location and movements of those entities in the augmented world is provided via the controller, that can request it from the federation services. The actual rules, that are required for a particular query, are provided by the controller, which itself retrieves them from the DL reasoner as stated above. Because the rules are not supplied in a syntax, that can be directly processed, they have to be transformed by the rule parser before it is possible to invoke them on the entities' spatial context.

RuleParser For each application query there might be a set of rules, which are retrieved by

the DL reasoner. Similar to the query parser converting queries, the rule parser transforms these given rules into an abstract syntax so that they can be cached by the controller and further processed by the rule invoker.

RuleInvoker The task of the rule invoker is to apply the rules given in abstract syntax to the spatial context of those entities appearing in an application's query. In order to execute the rules, the rule invoker can invoke certain functions in a common *function repository*. Each successful rule invocation results in higher-level spatial context, which is finally returned to the central controller, where it is also cached and forwarded to the persistence manager, before it is finally provided to those applications, that have subscribed to the corresponding query or actively issued the query.

Function Repository This repository contains an extensible and reusable set of *common reasoning functions* that support the processing of abstracted spatial queries. These functions can be addressed in the actual rules that correspond to certain queries. In general there are two different kinds of functions. Spatial context access functions simply return the values for a given property of an entity. For instance, there are different functions returning the unique identifier, class identifier, position, orientation, velocity and so forth for a queried entity.

The other kind of function is directly used for rule-based reasoning in order to deduce information. For example, the rule-based reasoner may process a rule, in which a function is used for deciding whether the time-to-collision (TTC) between two automobiles is less than fife seconds. If the repository contains such a function, then the reasoner can invoke it on virtual or real entities.

5.3.3.3.5. Persistence Manager

The persistence manager of the reasoning service can be compared to the one in SCORE's federation subsystem. It also handles the storage and retrieval of spatial context. However in this case it does not manage low-level context, but only implicit spatial context. That means, the reasoning service's persistence manager enables only the storage of those location-based information, that it is able to deduce.

Like the persistence manager of the federation service this one does not implement the actual persistent storage on its own. Hence it is dependent on an additional component, which not only provides the storage mechanism, but also enables the persistence management of higher-level spatial context.

5.3.3.3.6. SFS Query

This interface can be used by the controller to request explicit spatial context about entities appearing in the applications' queries. The reasoning service's need for low-level context matches the ability of one or more federation services. Therefore the controller is able to

query information from possibly multiple connected federation services.

5.3.3.3.7. Controller

The controller of the reasoning subsystem processes actively stated queries, (un)subscriptions to queries and the queries themselves, to which applications have subscribed. The workflows for subscribing and deleting subscriptions have already been explained. Now the workflow depicted in figure 40 on the next page shows the processing of a query, which is requested by an application via a method call at the query handler. The sequence of steps which is represented by this workflow is explained as follows.

In order to enable the reasoning about spatial context with respect to an application's query, the controller first looks up its query cache to search for the abstract syntax notation of a previously stated query, that equals the current one. In the case, that the query is not found in the cache, the controller uses the query parser to acquire a new abstract and processable representation of it. If historical implicit spatial context is necessary for processing the individual query, the controller can request it from the reasoning service's persistence manager.

The following step refers to the description logic-based reasoning of terminological context by means of the query in abstract notation and the historical implicit spatial context. However this is only necessary in the cases, that the current query has not been processed before, the ontological knowledge base of the terminological information has changed since the last equal query was processed, or in the case, that additional past implicit context is involved. Like the first two cased also the last one requires to invoke the DL-reasoner, because it could be possible to deduce new or additional information with the help of the historical spatial context. As its response, the DL reasoner returns a set of rules, that are processed by the rule-based reasoner later.

In general it is also necessary to query for current and for historical low-level context depending on the entities listed by means of the query language. Hence the controller accesses one or more spatial context federation services via the SFS query interface to obtain the required information. As follows, the controller searches its cache again in order to find the rules, which correspond to the issued query. Though the DL-reasoner might have been invoked and provided a set of rules, it could be possible, that these have been cached together with their abstract notation during a previous query was processed. If the controller does not find a matching rule or set of rules in its cache, it calls the rule parser of the rule-based reasoner in order to obtain the required abstract notation of the rules.

After the controller cached the query and the rules together with the abstract data representation of both, it provides the abstract query and the rules with the explicit spatial context to the rule invoker, that applies the rules to the entities' spatial context according to the query. The result of this invocation represents the interpreted higher-level spatial context, which the controller passes back to the query handler. The query handler itself finally returns it to the querying application thus completing the workflow.

In general this workflow is equal to the sequence of operations, that apply to the query processing for event-based notifications. The initiator “anApplication” has just to be exchanged with the subscription component for an application, and additionally a method call must be added from it to the event sender, which should dispatch the structured event. Due to this large similarity, a corresponding figure is omitted here.

The actual response of implicit spatial information is either returned to the calling query handler or to the subscription component, which pushes the information to the event sender. Therefore the according data flow depends on whether an application called an interface method or subscribed for a static query of spatial context.

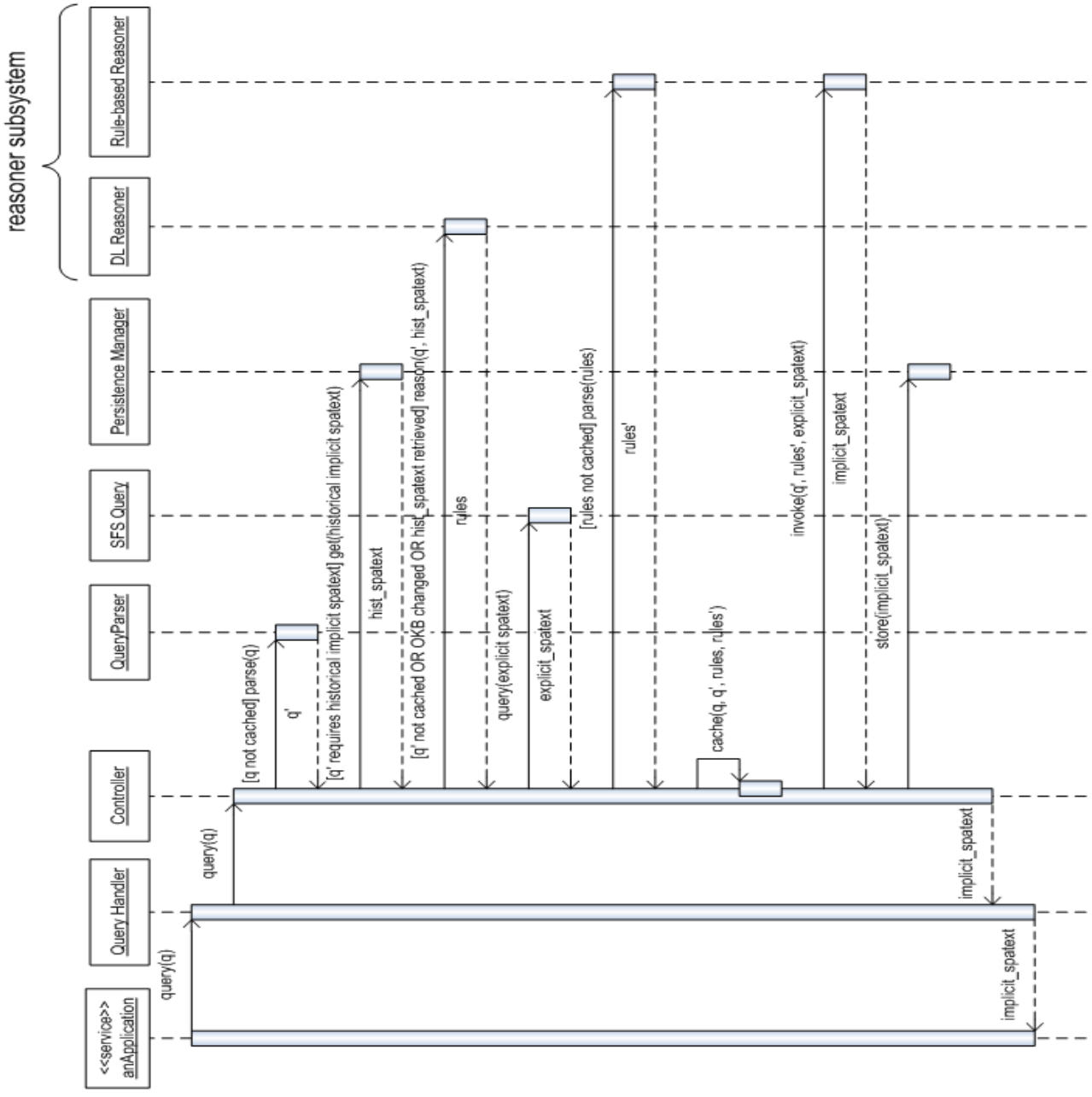


Figure 40: SRS workflow when processing a query (UML sequence diagram, “q” stands for query, “spatext” for spatial context and “OKB” for ontological knowledge base)

5.3.3.3.8. Design Rationale

In the previous sections the general design of SCORE's reasoning service has been presented. Its constituents are designed around the central controller component. Also the workflows for query processing, subscription and the removing of a subscription by applications are explained. This chapter is aimed at discussing the reasons, which are responsible for the design decisions.

General Design Similar to the broker of the federation service, the central controller of the reasoning service mediates between applications' requests and the other internal modules of this service. Therefore it is possible to change or extend the implementation of those components, and also exchange whole modules or add additional ones in order to increase the reasoning service's functionality. Like the component-based design of the federation subsystem also the design of the reasoning service allows for the consideration of the maintenance qualities *modifiability* and *extensibility*.

As an example, the exchange of the rule-based reasoner for another reasoner could possibly be considered in order to introduce a new syntax for the specification of rules or use a completely different approach for reasoning about dynamic spatial context. In this case neither the service interfaces, nor the query parser, the subscription handling component or the persistence manager have to be adapted at all. Also the controller does not have to be exchanged if its implementation separates the caching and parameter passing of the rule items from their particular data representation. This can be achieved by encapsulating the individual sets of rules and their abstract representation into data containers, where the controller solely has to know which container matches which query.

As seen before, there is a large similarity between a direct method invocation by applications with a query as the method parameter and the repeatedly occurring requests of subscription handling components, which also specify a query as parameter on behalf of applications. This allows for joining the controller's query methods in the later implementation. Originally two separate methods would have been required, where one serves the processing of queries, which are directly issued by applications, and the other would be responsible for the query processing on behalf of the subscriptions.

Like each federation service enables multiple reasoning services to connect to it, also the controller of each reasoning service is able to access multiple federation services via the SFS query interface. Therefore it is possible to dynamically create a “n-to-n” relationship between the instances of both services, which allows for a high *scalability*, both for multiple applications and multiple context providers.

Reasoner Subsystem The design decision for using external software for the description logic-based reasoner is simply based on the availability of those frameworks. In order to use a most flexible reasoning approach, I decided for modeling the general concepts of the augmented world by means of ontologies. Because of the common characteristics of ontology-based context reasoning (see chapter 2.4.2.1 starting on page 51), it is not too costly to inte-

grate an appropriate DL reasoner into SCORE. In contrast to this, the rule-based reasoner is aimed at being implemented during this thesis, because their is currently no suitable reasoner available, which is able to handle the links between static and dynamic explicit and implicit spatial context in combination with a very high efficiency of information inference such as it is required for Augmented Reality applications. The reason for using a rule-based approach is in principle derived from its discussion and comparison to machine-learning techniques as described in chapter 2.4.2.2. The main advantage is its suitability for efficiently reasoning about spatial context by means of common rules. These can be declared independent of persons' viewpoints because they have to take general physical laws into consideration.

The idea of a function repository represents a very important design decision. It allows to store an extensible set of common reasoning functions, that have to be specified in a way, so that they can be reused for different entity classes. This enables to build a consistent and reusable function pool for rule-based reasoning about spatial context. These functions will be further explained when discussing the combination of conventional spatial context models and contextual ontologies later.

Caching and Persistence Since queries, parts of the reasoner's results such as the reasoning about terminological knowledge and the retrieval of rules matching a query are often invariant, the controller has the additional task of caching this information in order to improve the processing performance for succeeding queries. Thus like the federation service's warehouse also the reasoning service provides a caching mechanism, which significantly increases the performance with respect to *response time* and *throughput*.

Also similar to the broker in the federation service the reasoning service's controller accesses a persistence manager to store higher-level spatial context after it has been interpreted and to retrieve it as past context data for further information deduction. The persistence manager hides the complexity and particular properties of the actually used persistence provider with the help of a common interface. The decision for this particular design is again based on the *portability* aspect, because in this case porting SCORE does not necessarily involve to port also the persistence provider.

5.3.3.4. Representation and Reasoning of Higher-Level Spatial Context

The idea behind SCORE's design is the uniform representation of general knowledge about the applications' domains. Here it is most important to know, that the entity classes, which have been frequently addressed so far, are not simply picked because they sound well or match the tasks of a particular application. In contrast, they obey a strict terminological declaration of common concepts of both the virtual and real world, which the developers of applications should agree on in order to guarantee a consistent and reusable information modeling. These terminologies are specified by Web resources, and are managed in a widely used Web-based ontological knowledge base, which can simply be accessed, modified and extended using a web browser.

With respect to the physical and virtual entities of an application, the important prerequisite for using SCORE is the matching of entity classes to ontological classes, entity class relations to ontological binary relations between classes, and entity class attributes to ontological properties, which assign data types to ontological classes.

That means, that all the appearances of an individual entity class, that is used in the context of SCORE for at least one application domain, are corresponding to each other. The same also applies to the binary relations between classes and the low-level spatial context attributes of entities, except for the entities' identifiers. These can be selected freely on the condition of their uniqueness.

This also means, that the developers of those services, which make use of SCORE by specifying entity classes and / or relations between them, have to address themselves to this kind of knowledge representation. This also includes both the developers of context provider services and applications. This is due to the fact, that the first have to declare the classes of the entities, which their services provide spatial context for, and the second want to use the federation and / or the reasoning services. Since the federation services aggregate the explicit spatial context of all available entities of exactly these classes, queries on the basis of them must use the corresponding class names. The same concern refers to the reasoning services, which provide their functionality on the basis of the query language. In addition to the optional query expressions regarding the entity classes, also the binary relation between its domain and range class accords to the ontological knowledge base.

Though it is possible to use specific namespaces for resources in order to declare one or more individual ontologies for a particular application, I suggest to stay with a global knowledge representation, which should be consistently extended instead.

Design Rationale The basic design idea I followed, is the declaration of only the basic terminology of the common management of spatial context. The rest should be the task of the service and application developers, which want to use SCORE's capabilities of context aggregation, federation and reasoning, and add their domain knowledge to a common ontological knowledge base (OKB). I intentionally decided for the previously mentioned restrictions which refer to the addressing of classes, their relations and attributes.

On the one hand it requires service developers using SCORE to have a look at the OKB. Though the current system design does not involve an adapted editor for manipulating ontologies, there are a number of those tools available[1], which can be used for this purpose, because SCORE's OKB is specified in the commonly used ontology description language OWL as described later. However it is also sufficient to use any text editor to change or extend the OKB, since its ontologies can be distributed among several rather small files.On the other hand it encourages developers to share their application domain knowledge. The most important benefit is the reusability of the declared terminologies in later spatial context-aware systems, because each project extends the OKB. Therefore developers do not have to start from scratch when designing the concepts of their application domains.

1 For instance the **S**ematic **W**eb **D**evelopment **E**nvironment (SWeDE) is a plugin for the Eclipse IDE, and provides an OWL editor and validator (http://owl-eclipse.projects.semwebcentral.org/).

Certainly it could be assumed, that the two subsystems of SCORE involve a lot of knowledge about the application domain. Exactly this is intended, because there is a strict separation between the knowledge of the application's domain and its logic, which SCORE supports, but is clearly not dependent on.

5.3.3.4.1. Modeling of Terminological Knowledge with OWL Ontologies

In the previous section it was stated, that the ontological knowledge base is declared by means of an ontology description language. For the representation of the terminological information OWL DL has been chosen, which is presented in chapter 4.1.2.

Rationale With respect to ontology-based reasoning, OWL is a very powerful XML-based language, which has been recommended by the W3C in February 2004. It is based on RDF's features of data and meta data modeling, and RDFS's capabilities of defining the corresponding vocabulary and constraints[1]. OWL extends this first approach of knowledge modeling by specifying formal semantics with the help of additional restrictions in the usage of RDF. Though the first designed version of the OKB does not use all the complexity of OWL DL, and can rather be categorized into its OWL Lite sub-language, the involvement of an appropriate description logic-based reasoner framework requires the selection of an appropriate OWL subset, which both enables a maximum of expressiveness while it simultaneously guarantees computability and decidability.

The decision for using OWL as the description language for ontologies not only allows for an uniform modeling of common knowledge about application domains. Furthermore it enables the sharing of these information among cooperating applications and also other context management systems. Because the extensible structure of an OWL knowledge representation enables developers and system architects to understand the incremental extension of the contained ontologies, the corresponding design goal for *evolvability* is another reason for selecting this ontology language.

Knowledge Modeling Figure 41 shows the graphical representation of the current OWL knowledge base, which contains a hierarchy of ontologies for describing terminological information in SCORE. These ontologies are not encapsulated in SCORE's reasoning service, but are stored in separate files somewhere in the file system or also on a Web server. The current OKB is listed in OWL notation in the appendix in chapter 10.3. Because the OKB consists of multiple ontologies, these have to be integrated into SCORE's *root ontology*. This ontology is called “SpatialThing” according to the root class of this ontology, which has the same name. SCORE's system design intends for automatically integrating all separately created ontologies into one global OKB by means of their import relationships. Therefore figure 41 does not show a separation into individual ontologies, but the current whole OKB.

1 RDF(S) and OWL are described in chapter 4.1.

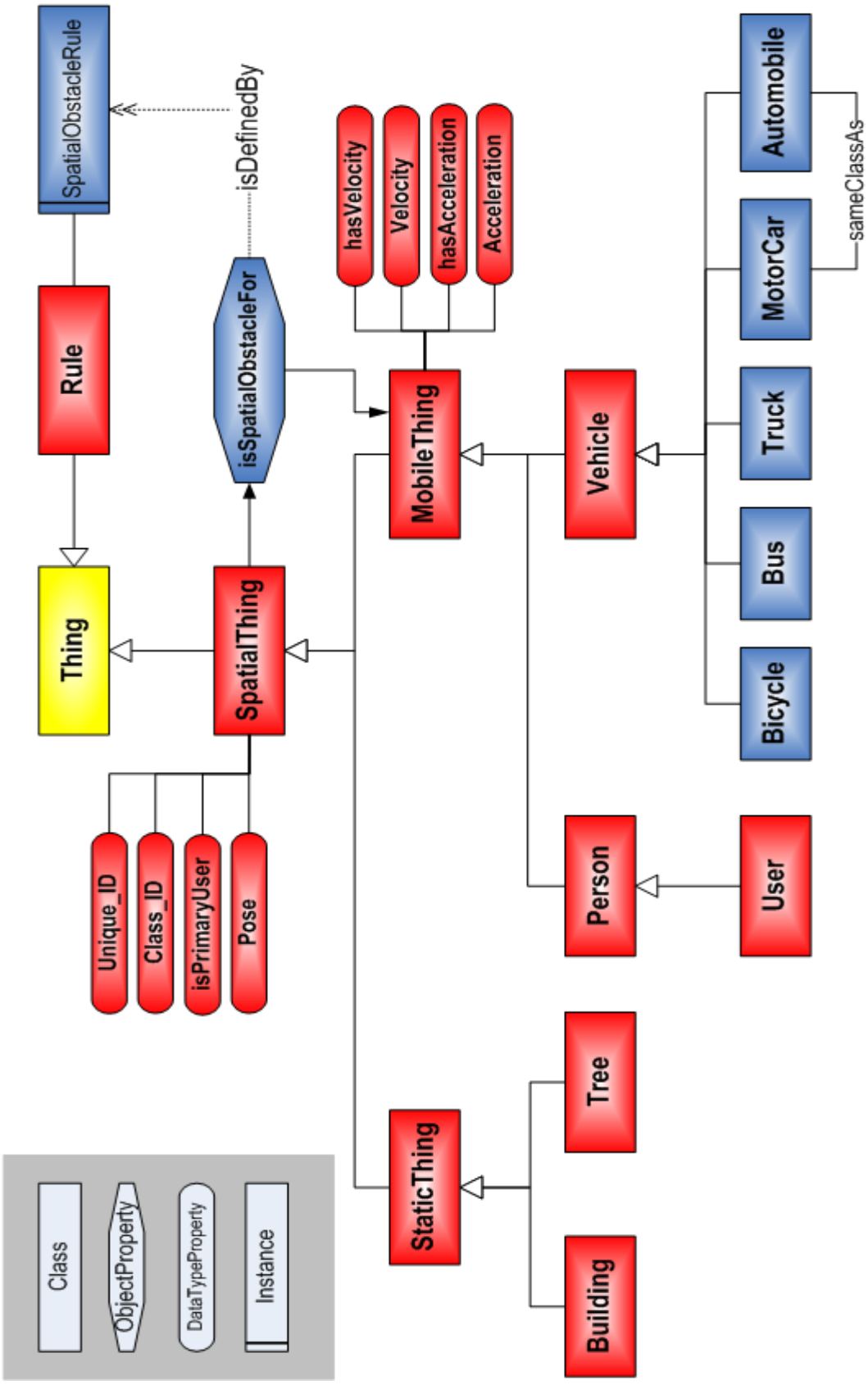


Figure 41: SCORE's basic OWL ontological modeling (red items) extended by the terminology of a first application (blue items)

The OWL class “thing”, which is drawn in yellow color, does not have to be explicitly declared, since all classes implicitly inherit from it. All the red items stand for the initial knowledge representation, which covers the basic class “SpatialThing”, its two subclasses “StaticThing” and “MobileThing” together with their common explicit location-based attributes, that are already know from the discussion of the low-level spatial context representation of the federation service (see chapter 5.3.2.3). The OWL class “Rule” is imported from a separate ontology, which I refer to as the *rule base*. Before addressing this ontology, first the subclasses of “SpatialThing” are presented.

Since SCORE is intended for the context management of spatial information, all newly modeled OWL classes, that describe the common properties of applications' entities, inherit from the “SpatialThing” class or from one of its descendants. This class represents the general properties of spatial things. That means that the classes of the applications' entities, which can be managed by the reasoning service, accord to “SpatialThing” or one of its extensible set of subclasses. Therefore each entity in the application domains denotes an individual of “SpatialThing” or its descendants, and is assigned a set of simple data types in XML Schema. These are the unique identifier, class identifier and the information whether an entity is the primary user. Additionally this class has a pose data property, which is modeled as a complex data type encapsulating position and orientation information in XML Schema (see also chapter 10.3.4 in the appendix).

“StaticThing”, which is the OWL class of all entities with fixed locations, and “MobileThing”, the class of all moving entities, are direct sub-classes of “SpatialThing”. Due to the dynamic properties of mobile things, the according class has additional data types at its disposal. These are the XML Schema simple types for velocity and acceleration together with the information, whether their values are valid. For this, “hasVelocity” and “hasAcceleration” are required, because an individual sensor system may sense an entity's location, but is not necessarily capable of also providing its velocity or acceleration data.

In addition to “SpatialThing” and its two subclasses, the initial modeling also provides a number of classes, which inherit from “StaticThing” and “MobileThing”. These are given as examples for how additional terminological application domain knowledge can be consistently integrated into SCORE's OKB. Figure 41 also shows several OWL classes and an object property, where all of these items are marked blue. They are emphasized in a different color, because they have been added during the design of a first application, that uses SCORE, and has also been developed in the scope of this thesis. Though this application will be presented in a later chapter, it is mentioned here to demonstrate how information, which was modeled for a previous application, can be reused to facilitate a current design process.

SCORE's first application is situated in the domain of driving assistance support for vehicles. For this domain a number of entity classes are required such as “Bicycle”, “Bus”, “MotorCar” and so forth. These classes have been declared to subclass from “Vehicle”, so that all the properties of this class are inherited. Therefore also the new classes can instantly be used by SCORE for reasoning about spatial context information of their entities. Also a binary relation, the object property “isSpatialObstacleFor” is added to the OKB. This property represents the knowledge, that an entity, which has spatial properties (“SpatialThing”) may become a spatial

obstacle for a mobile entity (“MobileThing”). Therefore a tree, a person or a bicycle could represent a spatial obstacle for an automobile. Those binary relations are important for the user-defined context reasoning, because higher-level spatial context about dynamic changing entity properties can be deduced on the basis of them.

Design Rationale The integration of the general concepts of the various application domains into a central OKB is enabled by extending the existing ontological information. The idea behind this approach addresses the requirement for supporting the sharing of terminological knowledge between cooperating applications and even between SCORE and other context management systems, which use OWL for the context representation (e.g. CoBrA, see chapter 3.2). It also allows for decreasing the cost of time when designing applications, that make use of SCORE, since each new application possibly increases the ontological information in the OKB, which can then later be reused by other developers.

It is also important to state, that the OKB does not necessarily consist of only one or two ontologies, which grow to an extent sooner or later, that will not be able to handle any more. This consideration is especially important, if developers use text editors in order to modify classes and properties in the OWL notation. In contrast it is possible to import ontologies into others, so that the constituents of the OKB are separated into clearly arranged parts in multiple files.

With respect to the design of the internal representation of the OKB, it is constructed by means of hierarchically arranged import statements in the ontology headers. The separation of the OKB into smaller parts allows for easily adding new ontologies or modifying the concept representation of existing ones. Therefore this important requirement addresses the maintenance qualities *modifiability* and *extensibility*. Because the OKB is modeled in OWL, a XML-based language, its evolution can be investigated by future developers. This consideration is based on the design goal of *evolvability.*

Another important design goal is *performance*. Therefore the reasoner subsystem of the reasoning service is responsible to provide an additional caching mechanism, that stores the OKB in main memory, so that its ontologies do not have to be reloaded each time when a query arrives. However it is important to dynamically detect changes to their persistent representation, which are stored in the local file system or on one or more Web servers. In this case the cached OKB must be instantly updated to a consistent state.

5.3.3.4.2. Combining Conventional Spatial Context Models and Ontologies

The federation services provide location-based information about entities by means of position, orientation and a number of additional basic spatial context attributes. However this information relates to a coordinate-based spatial context representation. Though the general relationships of entities have been explained by means of the modeled terminology for an application domain, the link between both worlds has not been addressed yet.

5.3.3.4.2.1. Rule Base

This yet missing link is provided by the *rule base*. It is a special ontology, which currently contains only one OWL class, i.e. “Rule”. This class has already been briefly addressed in the previous section. Each instance of this class represents a set of rules, which can be associated to one or more OWL object properties. This association is represented by the “isDefinedBy” statement, which can be declared for an OWL object property to denote, where the according rule resource is located (see example 7).

```
<owl:ObjectProperty rdf:ID="isSpatialObstacleFor">
  <rdfs:isDefinedBy rdf:resource="&rulebase;#SpatialObstacleRule" />
  <rdfs:domain rdf:resource="#SpatialThing" />
  <rdfs:range  rdf:resource="#MobileThing"  />
</owl:ObjectProperty>
```

Example 7: Linking an OWL object property to the corresponding rules

Following the link of the resource leads to an instance of the OWL rule class in the *rule base*. That instance actually defines this binary relation by means of the “Spatext Rule Language” (SRL), and represents the rules, which are used by the rule-based reasoner for user-defined reasoning about dynamic spatial context.

5.3.3.4.2.2. Rules defined in the Spatext Rule Language

The instances of the OWL rule class define the reasoning rules for their corresponding OWL object properties, which have an OWL domain and range class. Similar to this, also a rule instance has exactly one domain and one range variable (see figure 42), that can be later used in the definition of the actual derivation rules. These variables can be declared as any characters or character strings, which do not impair the parsing of the OWL document such as XML/OWL tags or not allowed special characters. For instance, when declaring rules for the first application simply “x” was used for the domain and “y” for the range variable.

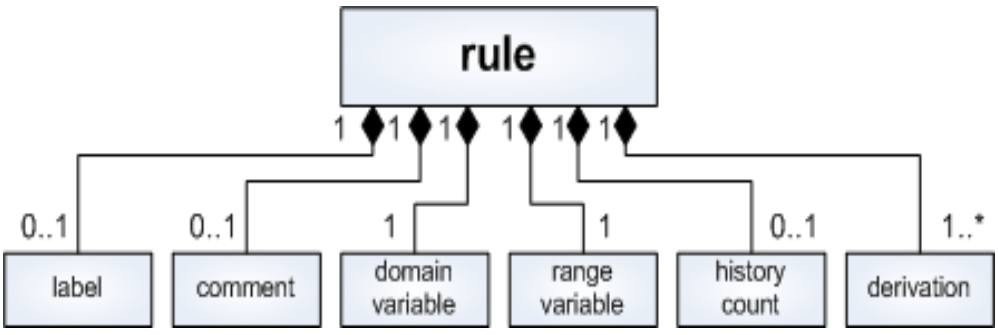


Figure 42: The structure of an instance of the OWL rule class in the rule base according to the Spatext Rule Language (UML class diagram)

Besides a label and a comment, the rule instance can also optionally define the “history count”, which is the number of distinct spatial context items of each entity at discrete moments in time. This concept has been explained when discussing the federation service and especially its broker component (see chapter 5.3.2.4.5).

For each rule at least one derivation has to be given within a RDF sequence. If multiple derivations are specified, then the rule-based reasoner will apply them in the given order. The derivations are designed to be similar to the “if” statements of general programming languages. Each derivation consists of an “if” tag with the attribute “case” that is used for specifying the actual derivation rule. For each “if” tag there is exactly one “return” tag, which denotes the return value for this derivation in the case it holds.

Example 8 shows an extract from those rules, which are defined for the “isSpatialObstacleFor” OWL object property with the domain “SpatialThing” and the range “MobileThing” (from example 7). The specified domain variable stands for all queried entities corresponding to the class, which is equal to or inherits from the OWL object property's domain class (“SpatialThing”). Of course this is the same for the property's range.

```
<Rule rdf:ID="SpatialObstacleRule">
  <rdfs:comment>the rules for the isSpatialObstacleFor OWL object property</rdfs:comment>
  <rdfs:label>SpatialObstacleRule</rdfs:label>
  <srl:variables srl:domain="x" srl:range="y"/>
  <srl:historyCount>100</srl:historyCount>
  <srl:derivations>
    <rdf:Seq>
      <rdf:li>
        <srl:if srl:case="angleBetween(x,y)<0.3 AND isInFrontOf(x, y)==1 AND
          {ttcBetween(y,x)<=10.0 AND ttcBetween(y,x)>5.5 OR distanceBetween(x,y)<=100}">
          <srl:return>information;front</srl:return>
        </srl:if>
      </rdf:li>
      <rdf:li>
        <srl:if srl:case="angleBetween(x,y)<0.3 AND isInFrontOf(x, y)==1 AND
            ttcBetween(y,x) <= 5.5 AND ttcBetween(y,x) > 2.0">
          <srl:return>warning;front</srl:return>
        </srl:if>
      </rdf:li>
...
    </rdf:Seq>
  </srl:derivations>
</Rule>
```

Example 8: An excerpt from the "SpatialObstacleRule" which is associated with the “isSpatialObstacleFor” OWL object property.

The actual definition of the derivations is done in the SRL “case” attributes of the “if” elements. In the condition of an “if-case-return” expression boolean operators (AND, OR, XOR), comparison operators with exactly two arguments (<, <=, ==, !=, >=, >) and *functions* can be used. It is also possible to provide nested expressions by using the “{“ and “}” brackets, and nest functions themselves if the return type of the nested function matches the parameter type of its surrounding function. This is shown in the following example.

```
{f1(f2(x,f3(y)),f4(x))!=7 XOR {f5()==true AND f1(x)>0}} OR f6(y)<5.7
```

Example 9: Spatial Context Rule Language: nesting expressions ("x", "y" denote the domain respectively range variable, and "f1" to "f6" are reasoning functions.

5.3.3.4.2.3. Reasoning Functions

All the *reasoning functions* (e.g. "angleBetween" or "ttcBetween"), which are declared in the derivations, have to be defined in the *function repository* in the reasoner subsystem. Besides functions, which can be employed to infer spatial context, there are also functions for retrieving the values of particular explicit spatial context attributes, that can be used with comparison operators or be passed as parameters for another function.

After the rule parser provided an abstract representation of the rules, the rule-based reasoner's rule invocation component calls the specified functions with all the stated parameters. If at least one parameter is given, then it can denote a simple value such as "true", "27.1", "abc" or also occurrences of a domain or range variable. In this case the variable is replaced by the explicit spatial context information of those entities, that match the corresponding domain respectively range of the property specified in the actual query. Because functions may be dependent on past context, they are provided a specified number of history items ("historyCount"), which also include the current data.

For example, with respect to the application's query

"`Vehicle=aVehicle isSpatialObstacleFor Automobile=myCar`"

the function "angleBetween" in the first derivation of the previous example 8 will be called with "x" replaced by the explicit spatial context history of the entity "aVehicle" (OWL class "Vehicle") and "y" replaced by the explicit spatial context history of the entity "myCar" (OWL class "Automobile"). However the rules would not be invoked at all, if the "isSpatialObstacleFor" object property did not exist for the given OWL classes or their corresponding super classes.

In the case, that the complete derivation condition resolves to be true, the rule-based reasoner returns the content of the corresponding "return" tag to the central controller, without further investigating the remaining derivation rules. For a more formal description of the syntax of the "Spatext Rule Language" the interested reader is referred to chapter 10.3.5 in the appendix, where it is defined in XML Schema.

5.3.3.4.2.4. Design Rationale

The rule base represents an extensible repository for defining knowledge about the dynamic properties of common entity classes. It enables ontology-based context reasoning about spatial information contained in traditional and highly efficient coordinate-based context models. Because OWL allows for object property inheritance, their associated rules also form a corresponding inheritance, which provides scope for additional performance tuning. Here the ob-

ject property inheritance can be used to find a sub-property of the one stated in the query, which has the same domain and range class or corresponding super classes of those in the query. If the more specialized object property involves a clearly simpler structure of derivations, then it can also be used for providing the same reasoning results. This is also very useful in the case that a subclass relationship between the specialized domain and range classes and the object properties domain and range as their according high-level super classes involves a lot of transitive reasoning steps.

For instance, the query "`person=* drives motorcycle=*`" could also be answered by reasoning about "`person=* rides motorcycle=*`" without having to visit the "vehicle" class as shown in figure 43.

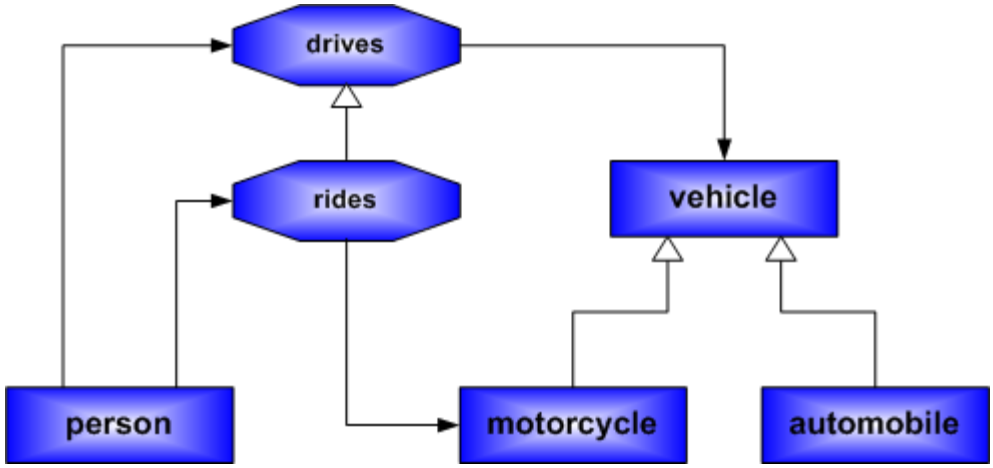


Figure 43: Reasoning about OWL sub-class and sub-property relationships

By separating the rules from the object properties they belong to, a clearly arranged and logically understandable structure is achieved. Certainly, rules can not only be easily modified and extended, but also reused and shared, since they represent instances of an OWL class. As a matter of fact the rule base follows the pattern of the ontological knowledge base by contributing to the design goals of *modifiability*, *extensibility* and *evolvability*. The latter quality attribute result from OWL's versioning capabilites. Therefore developers can investigate previous versions of the rule base in order to find out *what* and most important *why* rule definitions have been modified.

The "Spatial Context Rule Language" is a powerful language for describing rules for object properties. It enables the specification of nested boolean expressions, comparisons and also nested functions. Because there is a large similarity between the definitions of derivations and the "if" statements of programming languages, developers, which are new to SCORE, can instantly start to specify their own rules without having to learn a completely new and often very complex rule description syntax. Therefore the *usability* of the reasoner service for system architects and developers is increased a lot.

The higher-level dynamic spatial context, which is deduced by the rule-based reasoner follows

the user-defined approach, because it corresponds to the “return” statement of that derivation condition, which holds first. Therefore application developers can completely decide on their own, which information should be provided for particular rules. This represents a very flexible approach and allows for later extensions in subsequent projects, because also complex statements about concepts in the augmented world can be retrieved.

Similar to the ontologies, which describe the common concepts of the augmented world, also the rule-base ontology must be cached in main memory, so that the rules can be instantly applied without having to retrieve them first from the source, where they are persistently stored. The mirroring of this information is necessary for the design qualities *response-time* and *throughput*. If a system architect or a developer modifies the rule-base at run-time, the changes must be detected by the operating reasoning services and also applied to the local cache at once.

5.3.3.5. SpatextData as the Basis for Communicating implicit Spatial Context

“SpatextData” is solely used for the communication between a reasoning service and an application. As shown in figure 44 it contains the query result for the interpreted spatial context with respect to one entity whose class matches the query's object property's domain and one entity whose class matches the query's object property's range, where the spatial context of both entities corresponds to the whole query.

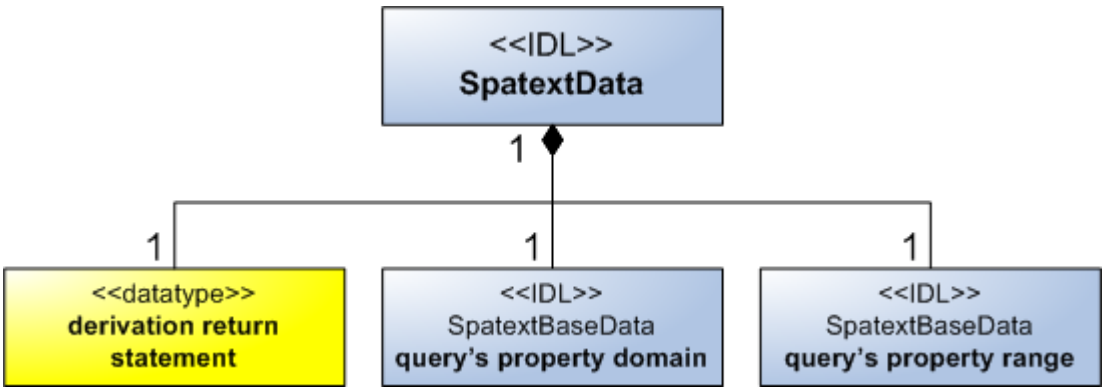


Figure 44: The constituents of "SpatextData" in UML notation

In addition to the higher-level spatial context represented by the derivation return statement of the rule matching the query's binary relation, also the explicit spatial context of both individual entities in its domain and range are returned.

Because of queries with many-to-many statements, applications are provided with collections of “SpatextData”, where each item corresponds to a particular relationship of two entities in the queried OWL property domain and range.

Design Rationale The implicit spatial context, which is sent to the querying application, corresponds to the “return” statement of the matching derivation. This simple but yet flexible approach standardizes the representation of spatial context for communication. Therefore application developers can foresee the possible query results by investigating the rule base, and adopt their applications to the possible responses for interpreted information in their application domain. The additional transmission of the explicit context data of the entities, which match the derivation rule with respect to the object property's domain and range classes, takes the application's requirement for processing low-level context into consideration. This information also contains the unique identifiers and class identifiers of those two entities, for which the given implicit context is valid.

5.4. Persistence and Access Control for Spatial Context Information

The compromise between development costs and functionality and the trade-off between development costs and both security and privacy, which are stated on page 98, results in the fact that the dynamic persistence management for and the access control to spatial context are not constituents of the design goals.

Of course the pre-defined terminological knowledge in SCORE's ontologies and the rule base is not lost after the reasoning subsystem is shut down. However the interpreted implicit spatial context of the reasoning service and the aggregated explicit context of the federation subsystem is dropped in this case, because the information is currently only cached in main memory.

Rationale The requirements for (spatial) context management involve the system's capability of storing not only static, but also dynamic context information in order to retrieve them for a later interpretation process. The current design allows for a persistent OKB, but it does not yet add the reasoned information to it.

Since the properties of explicit context differ considerably from implicit context information, I propose to also separate the persistence mechanisms for both kinds (see chapter 5.3.1 starting on page 100). The system design addresses this idea by the integration of persistence manager interfaces for both the federation and reasoning services. Additionally an appropriate spatial context model (“SpatextBaseData”, OWL ontologies) is already provided, which can be directly mapped to a suitable persistence provider. Therefore a later development of the missing persistence functionality is definitely not very difficult.

With respect to the selection of a persistence provider for managing explicit spatial context I propose to use an efficient relational database system providing dedicated extensions for spatial data (see chapter 2.4.1).

When searching for an appropriate persistence provider for SCORE's implicit spatial context, I refer to the capabilities of OWL. Here it is certainly a good idea to introduce new ontologies imported into the root ontology “SpatialThing”, that store reasoning results of dynamic higher-level spatial context. This is facilitated due to the OWL instance statement. For each query

with a given property, the matching entities are stored as OWL instances together with additional domain and range statements added to the corresponding OWL object property. Therefore it is possible to additionally use this ontology in future reasoning processes.

Access Control The user's privacy and the security of hers sensitive context is an important requirement for the acceptance of context-aware applications. The separation between terminological information and dynamic spatial context of entities allows for a facilitated integration of an access control mechanism in a later development process, because only the dynamic context has to be secured. The actual authorization is generally bases on rules, which define the access of security objects such as the explicit or implicit context of an entity for user, groups, applications or processes in the distributed environment. Here I would decide for the mandatory access control (MAC), because it restricts the access to each security object by means of an associated security label, which is still valid after a security object has been sent between services ([FK02], page 10). Thus a global access control can be implemented for both the federation and the reasoning service.

5.5. Hard- and Software required by SCORE

In order to prepare the implementation process, that follows from SCORE' design, also the supported hardware and the additional off-the-shelf software, which will be actually used, has to be taken into consideration.

Hardware Since SCORE is intended to provide its functionality to applications, that are aware of spatial context of entities in the augmented world, at least the federation service should be implemented a way, so that it is able to operate on mobile devices (handheld computers, sub-notebooks, etc.) or also stationary devices with comparatively low computing power such as embedded computer systems. Instances of the reasoning service can also be deployed on hardware within the user's environment. However the computing devices addressed by the reasoning subsystem should be rather efficient regarding their computing resources. This is due to the fact, that both the description logic-based and the rule-based reasoners certainly show an increased computing load.

A very important requirement is the fast communication between the reasoning and the federation services. The reasoner must continuously acquire low-level spatial context, which is provided by possibly multiple federation services. If there is only one instance of each service for an application, then a maximum efficiency is guaranteed, if both services and possibly also the application run on the same hardware platform.

Software SCORE uses an underlying *communication infrastructure*, which supports the resource discovery and location transparency of its services. Here it is possible to select any

service-oriented architecture, which supports a loosely coupling of services, and provides a mechanism to publish their interfaces so that their descriptions and semantics are available for a dynamic binding to context providers, applications and SCORE's two principal subsystems. Since DWARF, that is already available at the chair, represents such an architecture enabling to dynamically connect services by satisfying these requirements, this framework has been chosen as the current deployment platform for SCORE. However it is also possible to use the *Web Service* technology stack instead of DWARF in a later project. It provides means for finding services via UDDI[1] on basis of their service descriptions in WSDL[2]. The actual communication between services allows for XML-based SOAP[3] messages and also remote procedure calls via XML-RPC[4]. With respect to SOAP the transport protocol is independent of the message. Therefore HTTP, FTP or also SMTP could be used in this case for instance.

In order to obtain and deduce terminological information from the ontological knowledge base, an off-the-shelf *OWL reasoner* has to be selected. For this purpose the open-source Semantic Web framework Jena2 has been chosen. Though Jena2 restricts the programming language of SCORE's reasoning service to Java, it nevertheless has been selected, because it provides an easy to use OWL API and both a volatile and persistent storage mechanism. Above all the OWL API facilitates the retrieval and reasoning about the terminological information, that is contained in SCORE's ontologies. Also Jena's persistence support can be employed in a later project to store implicit spatial context.

5.6. Global Software Control

The global software control addresses the fundamental kinds of control flows in SCORE's two services. This consideration refers to the importance of using a multi-threading-based control for both subsystems.

Threads Like the controller in the reasoning service, the broker of the federation subsystem is a central component mediating the access to all the surrounding modules. In order to avoid bottlenecks and to increase overall performance especially in multi-processor environments, the broker or respectively the controller must provide support for multiple threads, that refer to the accessing components.

For example, different threads which are dynamically created on the calls of methods in SCORE's query interfaces try to simultaneously process the query by invoking the functionality of the respective service.

1 **U**niversal **D**escription, **D**iscovery and **I**ntegration is an XML-based protocol used for communication with Web Service registration brokers.
2 The **W**eb Service **D**escription **L**anguage defines the properties of a Web Service and how it can be used.
3 The **S**imple **O**bject **A**ccess **P**rotocol describes a message for a Web Service by means of an “envelope” containing an optional header and the message body.
4 XML-RPC uses HTTP POST for calling a Web Service's remote procedure and HTTP Response for retrieving the return message.

Detecting and Avoiding Deadlocks However the usage of threads involves the requirement for avoiding deadlocks, especially in the case when multiple threads try to simultaneously access and modify the same objects within the data structures, that are provided by the caching components. Here the global control flow must be implemented a way so that deadlocks will not arise or that they can be automatically detected and resolved. This is possible by maintaining and observing the state information for all competing threads and the objects, which are accessed via a mutual exclusion mechanism in order to guarantee their consistency.

Event-driven Software Control The event handler of the federation service must be able to continuously process input of explicit spatial context. Here for each arriving event a worker thread is created from a thread pool. Though a response does not have to be sent to the context provider due to the asynchronous communication property, the spatial context must be aggregated quickly in the warehouse to make it public to querying reasoning services and applications.

For the consideration of the benefits and limitations of the synchronous and asynchronous communication mechanisms in the respective cases of their usage, the reader is referred to the discussion of the design decisions in the chapters about the needs and abilities of the federation service (5.3.2.1 and 5.3.2.2) respectively the reasoning service (5.3.3.1 and 5.3.3.2).

5.7. Startup, Shutdown and Exceptions

This chapter describes the prerequisites, which have to be taken into consideration on startup and shutdown of SCORE's services. In addition also the necessary reactions, that apply to a situation, in which a service can not operate properly any more, are discussed here.

Startup In contrast to the federation services, that can just be started without paying attention to any further conditions, the reasoning services require the access to the ontologies, which provide the common concept knowledge about the augmented world, and the ontologies of the rule base. In addition also the access to the XML Schema definition of the “Spatext Rule Language” is required in order to apply the user-defined rules by the rule-based reasoner. Since all this information is cached, it is only needed on the startup of the reasoning service or in the case, that modification to those resources are detected. The necessary address information consists of one URI for the root ontology “SpatialThing” (all other ontologies including the rule base are imported dynamically), and one URI for the rule language description.

Therefore all of SCORE's ontology resources and the rule language resource must be available on the startup of the reasoning service. This may include a valid connection to the Internet in the case, that the resources are located on a distant Web server, and not in the local file system.

Shutdown Because currently the design goals do not cover a persistence mechanism, there is no specific shutdown handling. However in a later development process, all the cached information in the federation or reasoning services, which are about to be shut down, has to be stored persistently.

Exceptions If the Web server, that provides the ontologies and the XML Schema for the rule language, becomes unavailable after the completed startup of the reasoning service, there will be no support for detecting applied modifications to these resources until the server can be contacted again.

In the case of a serious failure of one of SCORE's services, the underlying distributed communication architecture is responsible for the detection of the service's unavailability, and should be able to automatically restart this service in the best case.

6. Implementation of SCORE

SCORE's system design, that is explained in the previous chapter, describes the general architecture and functionality of the “Spatial Context Ontology Reasoning Environment”. Above all, this chapter is dedicated to the developers, who want to extend SCORE's implementation in future projects. In order to help them quickly start with the development of their own extensions, the detailed class design of both the federation and the reasoning service, which followed from the general structure of these constituents, is presented in this chapter. With respect to the federation service its warehouse's indexing structure is explained in detail together with a complexity analysis of the access operations. I will also state additional remarks about the implementation and its current state for both services later in this chapter.

Before starting with the explanation of the federation service, there are some important general remarks about the structure of the source code repository and the approaches used for debugging and testing the services.

6.1. Preliminary Remarks

As described in the second chapter of SCORE's system design, DWARF has been selected as the underlying framework for dynamically connecting cooperating services, which communicate via CORBA method calls or event notifications. As a matter of fact the service interfaces of SCORE support CORBA-based communication.

To allow for a clearly arranged source code structure, the implementation-specific classes of the federation and the reasoning service are located in different directories of the code repository.

General Structure of the Source Code Repository In addition to the general separation of the source code directories for the federation and the reasoning service, also common classes, that do not contain application logic, are stored in different locations. Thus these classes, which contain general data structures and frequently used utility functionality, can be reused in other projects, which are based on the same programing language.

Finally classes, which cover the test cases of both services, are separated from the directories holding the application logic classes. This approach helps to keep the source code repository clearly arranged.

Debugging During implementation there is often the requirement for debugging the written source code in the case of errors, which can be recognized by means of a logging mechanism. In order to support both the logging and debugging, the open source log4j library[1] is used for Java, and both the DEBUGSTREAM macro[2] and an own implementation (“dprintf”) is employed for C++.

1 http://logging.apache.org/log4j/docs/index.html
2 http://wwwbruegge.in.tum.de/view/DWARF/DwarfLoggingTutorial

Testing For testing the logic and data structures of both service there are dedicated classes containing various test cases for unit and integration testing. In order to test the complete system, an additional service has been implemented. The “SpatextBaseDataProvider” is a lightweight DWARF service, that can be used for functional and performance testing of one or more instances of the federation and reasoning service. It emulates one or more context providers by sending events of low-level spatial context to the federation services, which in turn aggregate the information and provide it to the reasoning services. With the help of debugging and logging information, it is possible to test both subsystems and measure their individual performance.

6.2. Spatial Context Federation Service

This chapter presents the object design of the federation service together with the implementation and its current state.

6.2.1. Object Design

The object design addresses the yet missing concepts of small-grained objects for linking those parts of the architecture, that have been identified during SCORE's system design, into the actual implementation. Figure 45 shows the classes of the spatial context federation service. In order to present a clearly arranged diagram, the classes, which are employed by the warehouse, are presented in the succeeding figure.

SpatextFederationService This class, which is named after the federation service, is also the principal one. It is responsible for instantiating the one and only broker object from the “SpatextBroker” class. It also creates a query session object and the spatial context event handler by passing them the pointer to the broker.

SpatextBaseDataHandler is the federation service's handler for “SpatextBaseData” events (see chapter 10.2.1 in the appendix for the IDL declaration). During instantiation it obtains the memory address of the broker, which it can access on the arrival of a new event by dereferencing the stored pointer. Its only operation is to update the aggregated context in the warehouse, that is hidden by the broker.

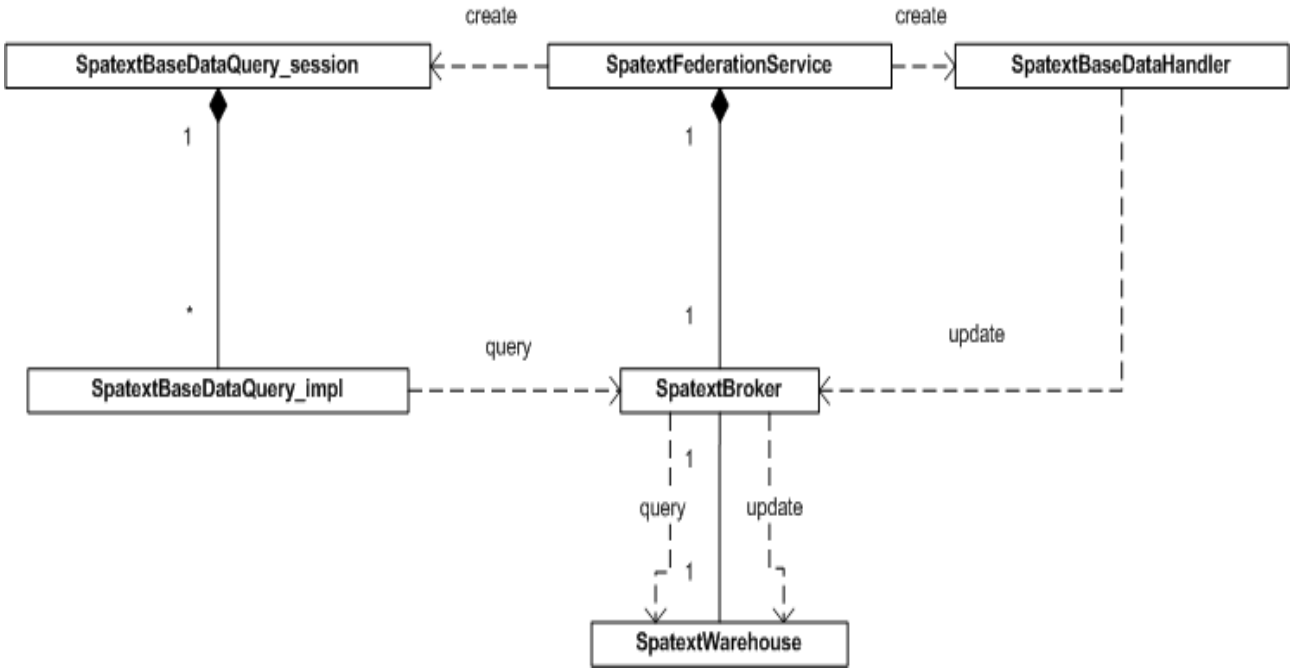


Figure 45: UML diagram of the current federation service's classes (dependencies denote <<call>> relationships, the hidden classes used by the "SpatextWarehouse" are shown in the next figure)

SpatextBaseDataQuery_session is responsible for creating exactly one object of "SpatextBaseDataQuery_impl" for each connected service, which accesses the federation service by method calls.

SpatextBaseDataQuery_impl implements the "SpatextBaseDataQuery" interface, that is declared in the according IDL (see chapter 10.1.1 in the appendix). Therefore it defines all the query methods for aggregated low-level spatial context information, which are declared there. Depending on the particular method, objects of this class (one for each application) return "SpatextBaseData", sequences or also matrices of it.

SpatextBroker The singleton spatial context broker consists of exactly one warehouse, and mediates the concurrent access to it, by forwarding the various query and the update calls. Because of its central position and since there is only one instance of this class per service, it is well-suited for accessing the persistence management system in a later development.

SpatextWarehouse There is also exactly one warehouse per service, which is employed for in-memory storage of the provided spatial context. It responds to concurrent query and update calls by threads, which access the warehouse via the broker. In order to process queries as fast as possible an efficient indexing mechanism is used. Figure 46 shows the additional classes, which are required by the warehouse.

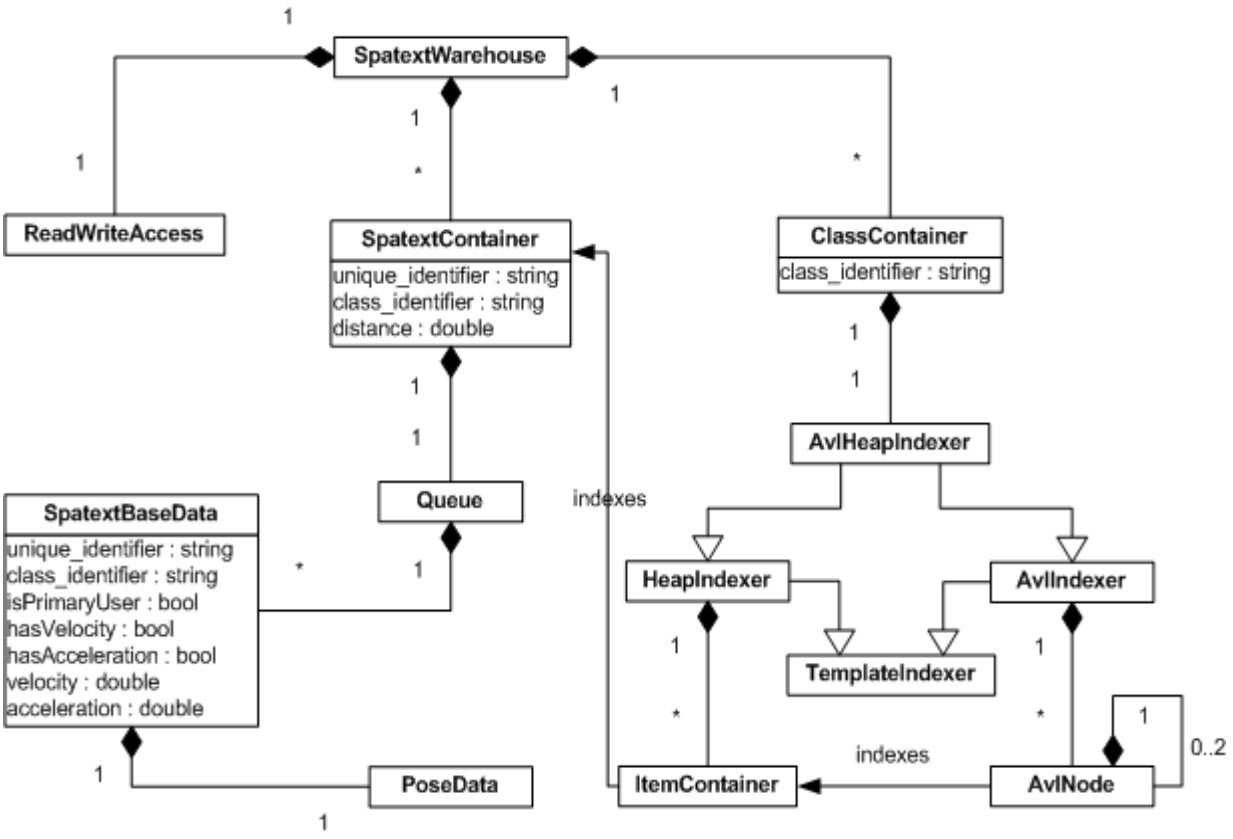


Figure 46: UML diagram of the classes used by the federation service's warehouse

ReadWriteAccess is developed as a reusable class, which encapsulates the common mutual exclusion logic for accessing objects. In this case multiple read operations, which occur simultaneously without a write operation, are allowed, whereas a write enjoys preferential treatment since it has to wait for all read operations to complete. This class is used by the warehouse to guarantee consistency on the spatial context data structure and its indexing mechanism.

SpatextContainer A special array in the warehouse holds pointers to “SpatextContainers”, which are not sorted within the array with respect to any of their dereferenced attributes. There are never two items in the array pointing to the same container. Therefore all containers are distinct. Each of the containers stores the unique name and class identifier of exactly one entity in the augmented world. The “distance” attribute represents the distance to the primary user entity. This attribute is only modified, if the primary user or the entity, which is denoted by this container, is updated. Each container also consists of a queue, that stores the current and past explicit spatial context of this entity.

The warehouse has additionally two special pointers at its disposal. One points to the container of the primary user entity, and one to the entity, which is closest to the primary user.

Queue This class is also one of the new reusable classes. The “SpatextContainers” use it to store pointers to “SpatextBaseData” objects, which represent the entity's present and past spatial context.

SpatextBaseData corresponds to the event, which is provided by the “SpatextBaseData-Handler” via the central broker. It is already presented in the system design chapter 5.3.2.3 starting on page 107.

PoseData mainly holds the entity's position and orientation values together with the timestamp, that is assigned to a “SpatextBaseData” event when it is created by the context provider.

ClassContainer The warehouse uses an array to store all distinct identifiers of all entity classes together with a pointer to an “AvlHeapIndexer”. The corresponding indexing data structure will be explained in detail in the succeeding chapter. Here the classes will be presented, which are used for indexing the low-level spatial context of entities.

Indexer Classes All of these classes are also designed to be reused in other projects by selecting one of the more specialized classes or composing the indexers to a complex structure.

The *“AvlHeapIndexer”* uses multiple inheritance with respect to the *“HeapIndexer”* and the *“AvlIndexer”* class, which both subclass from the abstract *“TemplateIndexer”* class. Here the AVL-based indexer implements an AVL-tree[1] of *“AvlNode”* objects, which are used for indexing the item containers of the heap indexer based on the unique identifiers of the entities that are in turn indexed by the heap. The heap's item containers are managed as a collection corresponding to an array-based approach of representing a heap. Its advantage is the level-based traversal of all heap items by simply iterating over the array. The heap itself is responsible for indexing all “SpatextContainers” belonging to the class in which the pointer to this *“AvlHeapIndexer”* is stored. In this case each (minimum) heap is arranged by means of the distance attribute of the indexed entity's container. To achieve this, each *“ItemContainer”* of the heap holds a pointer to one of the “SpatextContainers”.

6.2.2. Indexing of Low-Level Spatial Context

Among other classes the previous section also introduced those, which are responsible for maintaining the indexing structure of the “SpatextWarehouse”. For supporting a maximum of efficiency for query processing, a complex indexing mechanism has been designed for the warehouse, which is shown in figure 47.

1 The AVL-tree is named according to its inventors **A**del'son-**V**elskii and **L**andis.

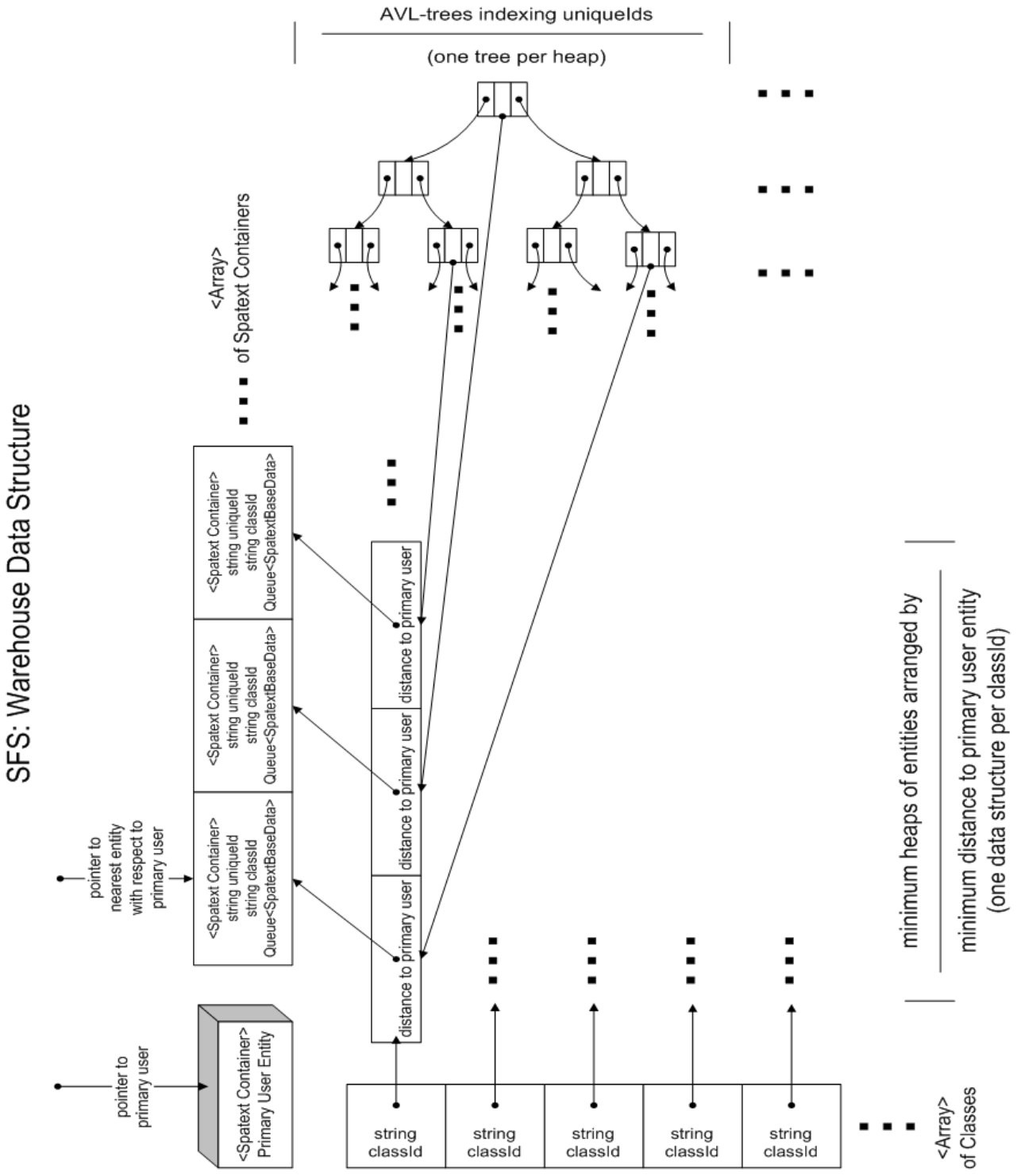


Figure 47: The design of the warehouse data structure in the Spatial Context Federation Service

The data structure of the warehouse has to be very efficient to handle high frequencies of "SpatextBaseData" events and of queries by the reasoning subsystem and applications. This is especially required for the domain of Augmented Reality, where information must be aggregated in "real-time". In order to explain the concepts of the data structure let us start with the spatial context container.

SpatextContainer Each individual entity is assigned a “SpatextContainer”, which is created, if the event handler provides a “SpatextBaseData” item, that corresponds to an entity that is not yet contained in the data structure. The spatial context container stores information about its entity's unique identifier, its class membership and the *queue* which manages a configurable number of “SpatextBaseData” items representing the spatial information history of this entity together with its most up-to-date context data. If a new “SpatextBaseData” item is poked to the *queue* while it is already full, then the oldest item is first erased from it.

The containers of all entities are arranged as an array, which does not necessarily specify any ordering on its items.

Primary User As explained during the discussion of the “SpatextBaseData” data type, the *primary user* is a special entity in SCORE. Like the other entities it is also assigned a “SpatextContainer”, that maintains its spatial history, and is shown as a block in figure 47 to distinguish it from the rest of the containers. In order to access the spatial information of the primary user in constant time, a special pointer in the warehouse marks its position in the “SpatextContainer” array.

Class Array On its own the “SpatextContainer” array does not represent an efficient data structure regarding the retrieval of spatial context items. Therefore a combined indexing approach is used to find entities and their context information fast. The foundation of the indexing mechanism is the *class array*, which contains the sorted distinct class identifiers of all entities. Besides such a class identifier, each item of the array also provides a pointer to a minimum binary heap and a pointer to an AVL-tree.

Minimum Binary Heap Each item in the class array is assigned an individual minimum heap, which is implemented as an array [KOS04]. Each item points to a “SpatextContainer” and is arranged by the *distance* between the entity represented by this container and the primary user. The set of all “SpatextContainers” addressed by all pointers of the items of all heaps contains only distinct members. That means, that an entity is indexed only once within all heaps.

AVL-Tree Per each class array item respectively per each heap there is one associated AVL-tree. Each AVL-tree is used for indexing the items of the corresponding heap by means of the *unique identifiers* of the entities, which are indexed by the heap. The AVL-tree is implemented as an internal binary search tree. That means the nodes containing the actual data, i.e. a pointer to a heap's “ItemContainer”, are not restricted to be leafs in contrast to external search trees. Therefore there is the probability that the correct node can be found without having to search the complete path down to the leaf.

6.2.2.1. Rationale

The “SpatextContainer” encapsulates the contextual history of each entity, which decreases the complexity for the implementation since it allows the separation of the spatial context from its indexing. That also means that when modifying the attributes for location-based information in the “SpatextBaseData” structure, it does does not impair the indexing data structure at all.

The history items of each entity are stored in a queue of configurable size. If the according container is known and if the queue itself is implemented on the basis of pointers, adding a “SpatextBaseData” item is possible in constant time. On the same prerequisites the runtime for returning ***n*** items of the queue by using the pop operator (the item is simultaneously deleted) or a stepping peek operators (the item is retained) is linear in ***n***. Also the deletion of items is efficient (pop ***n*** times).

The heaps can be used to find entities fast, if it is known, that they are are located rather close to the primary user. However this is based an the assumption, that the heap item, which points to the searched entity, is located close to the root node, that may not be the case in general. On the other hand, regarding a certain class, the retrieval of the entity, which is closest to the primary user, is possible in constant time if the class has already been found, since it is addressed by the root node of the heap. If a primary user does not exist, then the AVL-tree can be used for finding an entity in logarithmic time with respect to the number of items in the tree, if the corresponding class item has been found in the class array.

The class array itself is not assumed to become large, because generally there is only a rather small number of common classes in an application domain. Even if there are hundreds of entity classes (e.g. for describing the breeds of dogs) the runtime for finding a certain class is efficient when using a fast sorting and selection approach.

The presented design of the indexing data structure contributes to the fulfillment of the performance-related design goals *response time* and *throughput*.

6.2.2.2. Complexity Analysis for Access Operations

The worst case complexity of *insert*, *find*, *update* and *delete* operations regarding the whole data structure is stated as follows.

Insert If the spatial context of a new entity is provided by the broker, then the entity's representation has to be created and indexed in the data structure. The complexity analysis assumes, that it is known, that the item does not already exists. The total runtime is the sum of the following operations[1].

1. A new “SpatextContainer” is created and added to the container array (constant time). This also includes the initialization of a new history queue.

1 The stated time complexities refer to the number of items in the class array, heap or AVL-tree depending on the context where the complexity appears.

2. The entity's class is added to the class array if it does not exist yet. This can be done in logarithmic time using binary search if the array is kept sorted.
3. On the basis of the distance calculation between the inserted entity and the primary user a new item pointing to the created “SpatextContainer” is added to that heap, which is referenced by the corresponding class. This can be done in logarithmic time in the number of heap items, since the heap is a complete binary tree ([KOS04], chapter 2.3.1). The distance calculation runs in constant time, because the primary user entity can be found by means of a global pointer.
4. Also a new item is added to the according AVL-tree, which indexes the just created heap item. The complexity corresponds to the height of the tree, i.e. O (log ***n***) where ***n*** is the number of leaves in the tree respectively the number of all items which point to any heap items [MAY99].

Find There are two major possibilities for finding entities dependent on whether or not a primary user entity exists. If it is not contained in the repository then the minimum heaps can not be used efficiently, since the values regarding the distance to the primary user are not set. That means there is no knowledge about the distance relation with the primary user entity. However an entity's history can also be efficiently obtained with the help of the AVL-trees.

In the case, that the primary user exists, the find operation can be based on the heap representation assumed that the searched entity is located close to the primary user, or alternatively on the corresponding AVL-tree if a guaranteed logarithmic runtime is desirable.

In all cases the entity's class item must be found. This is possible in logarithmic time (binary search), if the class array is sorted. Searching the entity via the class item's heap accords to the following two steps.

1. Run through the heap level by level to find the heap item which points to the searched entity. This procedure compares to the linear examination of each heap item, if the heap is implemented as an array. Though the worst case run time is therefore linear, there is a good chance to find the entity fast, if it is located close to the primary user. In the case, that a query requests the class entity, which is located closest to the primary user, the root of the heap can be found in constant time.
2. Follow the link from the found heap item to the “SpatextContainer” that holds the location-based information of the searched entity (constant time).

Using the AVL-tree, which is linked in the already found item of the class array, results in logarithmic time for searching the entity.

1. By traversing the AVL-tree search the tree item, which points to the heap item linked with the searched entity. This can be done in logarithmic time [MAY99].
2. Follow the link to the heap item (constant time).
3. Follow the link to the “SpatextContainer” (constant time).

When the “SpatextContainer” is finally found using one of the these approaches, a number of contextual history items can be peeked in the queue in linear time.

Update An update of an entity's spatial context means the poking of the update information (“SpatextBaseData”) to the entity's historical queue. The update operation is similar to the find operation. After the according “SpatextContainer” is found, its spatial context information is enqueued. The operation on the queue is possible in constant time. Finally the according distance value in the heap item, which indexes the entity's container has to be updated (constant time).

Delete First, the “SpatextContainer” of the entity, which should be deleted from the data structure, has to be searched via the find operation. The corresponding indexing items in the class array, heap and AVL-tree are remembered during the search.

1. Erase the “SpatextContainer” together with its contents in linear time (***n*** pop operations on the queue, where ***n*** denotes the queue size).
2. Erase the remembered heap item ***I*** in logarithmic time as follows. Swapping ***I*** with the heap's root ***R*** results in ***I'*** and ***R***', where ***R***' is the new root item. Restore the heap condition with respect to ***I'***, and then delete ***R***' by moving the heap's leaf with the greatest distance value to the root and overwriting it. Finally the heap condition regarding the new root has to be restored.
3. The deletion of the remembered item in the AVL-tree is also possible in logarithmic time.
4. If the heap or the AVL-tree is empty then it means that there is no more entity for the corresponding class. Therefore also the remembered item in the class array can be deleted (constant time).

Since the primary user entity is managed centrally, the find and update operations can be performed in constant time. If it is deleted, then the distance values, which are stored in the items of all heaps, are implicitly invalidated. This means, they do not have to be explicitly set to a certain value, but are nevertheless no more used for finding entities since the find operation checks the existence of the primary user first before considering the usage of the search capabilities of the heap indexing mechanism.

If a new primary user is set, the distances between it and all stored entities have to be calculated in order to guarantee consistence. This is done by a complete traversal of the structure of all employed heaps (linear in the number of all entities, which are not equal to the primary user). This case is considered to occur rather seldom.

Certainly is is not possible to design all access methods to be most efficient due to necessary compromises. As described in the analysis of the operations' complexities, the *find* operation is designed to be the most efficient one, whereas the *delete* operation has to process the entity's history queue, the class array, its heap indexing structure and the AVL-tree. The

reason for this trade-off is the fact, that *find* operations occur much more frequently as requests for deleting an entity.

6.2.3. Implementation

This section contains important remarks which regard the implementation of the federation service.

Programming Language As one might have already guessed before, when the classes in figure 46 on page 145 have been presented, the federation service is implemented in C++ (the “AvlHeapIndexer” uses multiple inheritance, which is possible in C++). The decision for this programing language is based on the requirement of a very efficient data and indexing structure for the service's warehouse. For providing as much performance as possible, C-specific methods have been used for the main memory management in parts of the service (“malloc”, “free”) in order to additionally increase the speed of memory (de)allocation.

The portability aspect has been taken into consideration by keeping to the ANSI/ISO C++ standardization and only using programming libraries, which are compatible with commonly used operating systems. Thus the porting of the federation service, which was developed on a windows platform and deployed on Linux, is rather facilitated.

Template Classes The “Queue” class and all of the indexer classes including the “ItemContainer” and “AvlNode” are implemented as reusable template classes. They can handle various data types, which are specified at development time. Their special advantage is their support for very high performance, because they do not imply any costly type casts at runtime, but nevertheless allow for static type checking at build-time. For more information about the template concept the interested reader is referred to [STR92].

6.2.4. State of Implementation

Except for the complete persistence management of explicit spatial context, the federation service is implemented as described. Although the implementation of a persistence mechanism is not a design goal in this thesis, it is not too difficult to add the missing functionality to a new persistence manager class, which can be accessed by the central broker to store and retrieve context data.

6.2.5. Testing

The data and indexing structure of the federation service's warehouse is tested by means of the MS Visual Studio project “SFS_test”, which sets up various test cases. The integration test for the whole federation subsystem has been applied with the help of the additionally implemented “SpatextBaseDataProvider” service.

6.3. Spatial Context Reasoning Service

The sections within this chapter describe the detailed internal design of the reasoning service followed by remarks about the implementation.

6.3.1. Object Design

With respect to the reasoning service the object design additionally includes the integration of the Jena2 Semantic Web framework. Before investigating the classes, which encapsulate the application logic, the container classes are first presented, because objects of these are used to formalize the applications' queries and the retrieved rules.

6.3.1.1. Container Classes

The following figure shows the four container classes, which are designed for SCORE's reasoning service.

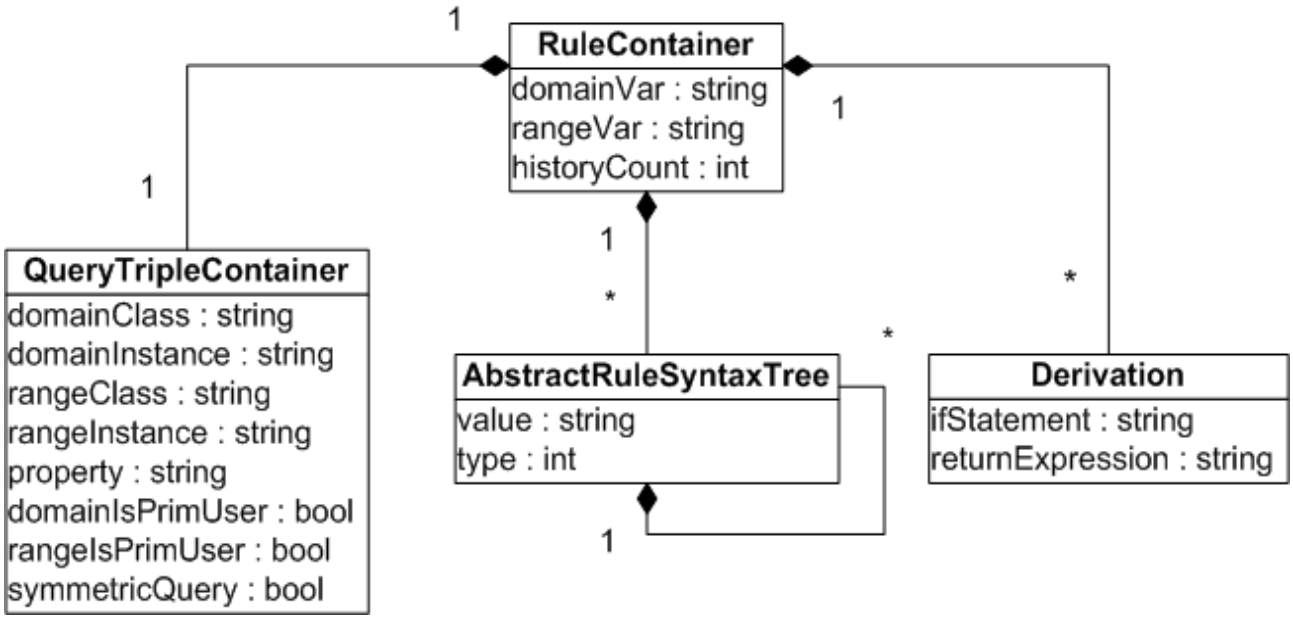


Figure 48: UML diagram showing the reasoning service's container classes

RuleContainer Per each application query there is one rule container involved. Its members are set during the complete reasoning process until the rules are actually applied to the explicit spatial context of those entities, which match the query. The rule container is composed of one query triple container and possibly multiple abstract rule-syntax trees and derivations. In this case, the number of the trees and derivations equals always.

The domain and range variable as well as the history count will be set by the DL reasoner when it obtains the matching rule set from the rule base ontologies. Therefore the variables

and the history count refer to the domain and range variable respectively the value of the “historyCount” element that is specified in the “*Spatext Rule Language*” (SRL, its declaration can be found on page 132 in the system design chapter).

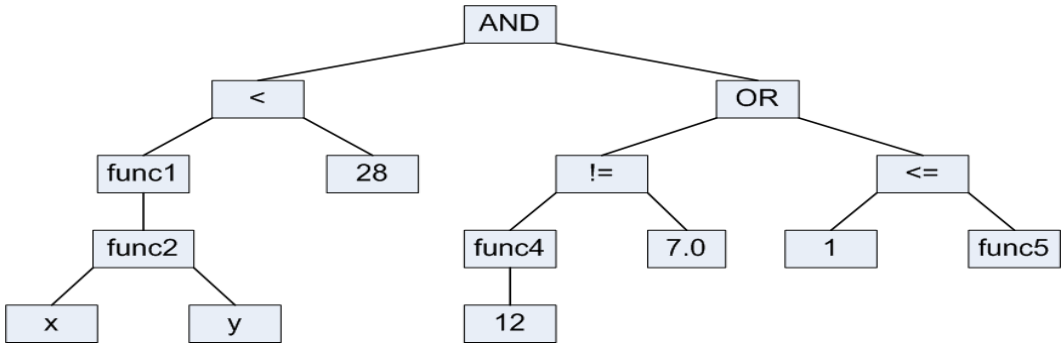
QueryTripleContainer holds the query triple, that is mainly composed of the domain and range class and the instances of both. Thus this container refers to the query language, which is presented in the system design chapter 5.3.3.2.1 on page 115. The “symmetricQuery” attribute is set true in the case that the entities of the domain and range class are equivalent. Because this property has to be checked frequently, it increases performance to compare the according strings only once.

Derivation Each derivation refers to one element of the set of rules, that matches the application's query and is retrieved from the rule base. A derivation object both contains the rule's condition (“ifStatement”) and its conclusion (“returnExpression”).

AbstractRuleSyntaxTree This tree is the result of the rule parsing process, which creates an abstract rule-syntax tree from the corresponding derivation object. Each node of this tree is a tree itself, and has none, one or more nodes as children. A node contains a value representing one of the rule condition's parsed items, that further can not be taken to pieces. A value's type corresponds to an integer of a pre-defined list, whose distinct items refer to the possible operators and elements of the rule language (e.g. “XOR”, “LESS_EQUAL”, “DOMAIN_VARIABLE”, “FUNCTION”, “LITERAL”, etc.). This approach is used due to the fact, that string comparisons are extremely costly in contrast to integer comparisons.

Example 10 shows an abstract rule-syntax tree for the following condition (“x” and “y” may represent domain or range variables).

```
func1(func2(x, y)) < 28 AND {func4(12) != 7.0 OR 1 <= func5()}
```



Example 10: Abstract rule-syntax tree for an example derivation condition

6.3.1.2. Classes providing the Logic

Figure 49 shows the class structure of this service except for the container classes which have just been explained. The classes in this figure are described by means of the principal control and data flow, which is not as obvious as for the federation service.

SpatextReasoningService is the class containing the “main” method. It is responsible for creating the “QuerySessionExporter” and the central controller.

QuerySessionExporter maintains one instance of the “SpatextQuery_impl” class for each querying application.

SpatextQuery_impl implements the “SpatextQuery” IDL interface (see chapter 10.1.2 in the appendix), which provides the query method for requesting implicit spatial context from the reasoning service. Each call of this method is forwarded to the controller.

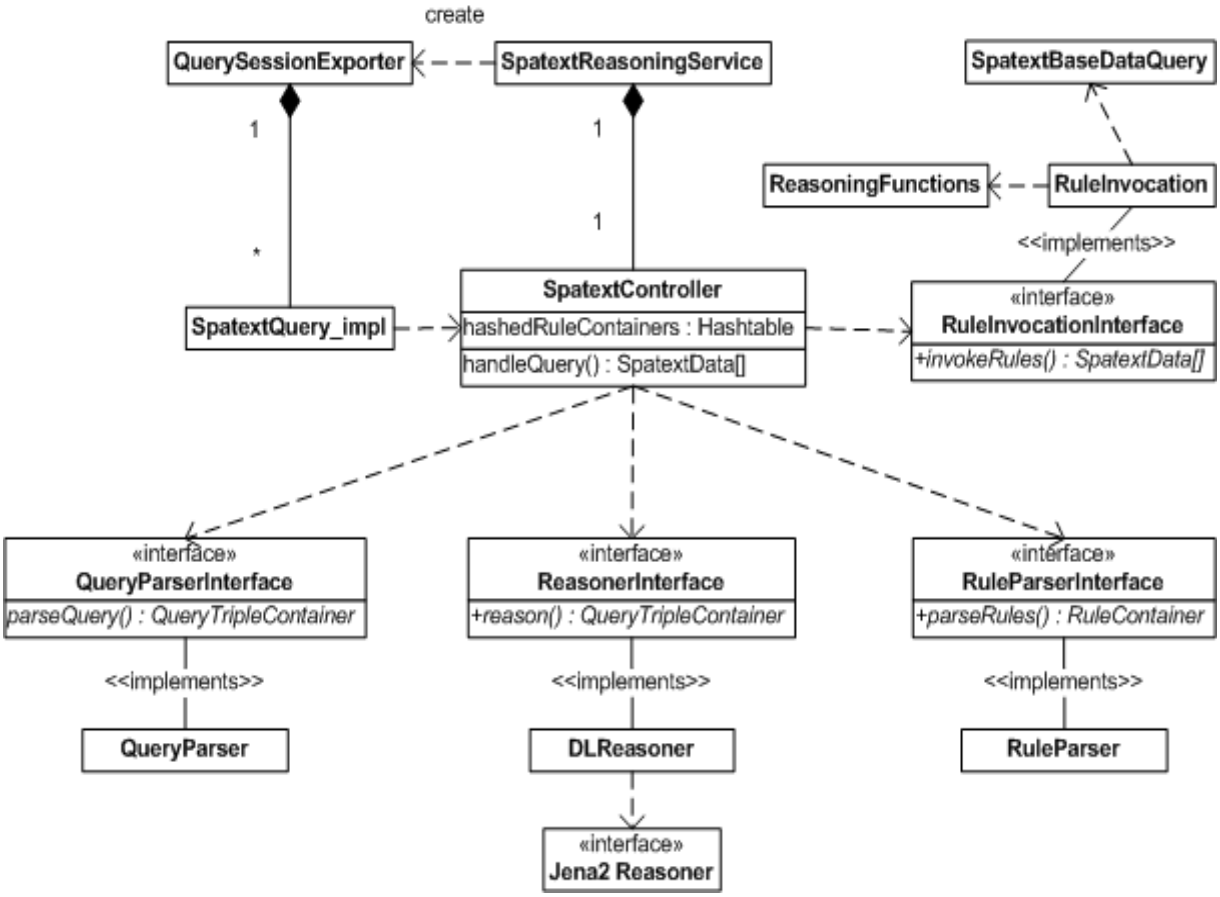


Figure 49: UML diagram of the current reasoning service's classes (dependencies denote <<call>> relationships, the hidden container classes are shown in figure 48)

SpatextController creates the query and rule parser, the DL reasoner and the rule invocation object on service startup. In the case of the DL reasoner, the complete ontology hierarchy including the rule base is loaded into main memory for faster access.

The contoller handles the incoming queries from the applications by applying the following steps:

1. The controller looks up its *hash table*, which contains valid rule containers (the hashed objects) of previously processed queries (the hash table keys), in order to find a container, that matches this query. In the case, that such a container can be found the controller retrieves it and proceeds with step six.
2. If there is no such rule container, the current thread, which accesses the controller, has to first parse the query string. Therefore it calls the *query parser* to obtain a "QueryTripleContainer" storing the abstract syntax of the query.
3. This container is next passed to the "*DLReasoner"*, which analyzes the abstract query by investigating the OKB, fetches the correct rules and creates a new rule container. Both the given query triple container and the *derivation* rules (conditions and conclusions) are stored in this rule container, which is finally returned to the controller.
4. Next the "RuleParser" is called with the newly obtained rule container. For each of the derivations' "*ifStatements*" a new abstract rule syntax tree is recursively built and added to the given rule container. After all rules have been parsed, the modified rule container is returned.
5. The now completely constructed rule container is stored in the controller's hash table with the application's query as the key.
6. The rule container is finally processed by the "RuleInvocation" class, that recursively analyzes the abstract rule-syntax trees and applies the conditions to the query.

QueryParser takes a query string and creates a query triple container, where it stores the extracted strings of the queried OWL domain and range class, the entity names/URIs and the OWL object property. Furthermore it checks whether the "as primary user" statement appears in the query, and sets the boolean values according to it.

DL Reasoner uses the *Jena2 OWL reasoner* to process the following steps with respect to a parsed query:

1. All URI's of the domain and range class and the object property in the query, which refer to OWL resources are resolved. That also means, that these items of the query are replaced by their fully qualified addresses, if they are stated by means of unqualified names.
2. The query's classes and object property are verified by checking whether they exist in the OKB, and can also be applied to the OWL object property. This validation allows for investigating transitive subclass relationships.

3. The correct derivations consisting of the values of the SRL “case” attributes and the corresponding SRL “return” elements are obtained by fetching the rule instance, which defines the OWL object property (“isDefinedBy” statement). These are stored together with the query triple in a new rule container, that is the return value of this class.

As shown in figure 49 on page 155, the reasoner subsystem, which was identified in the system design phase, is replaced by yet smaller subsystems. The reason is, that it should be possible to exchange these for other implementations on the basis of medium-grain components.

RuleParser takes a rule container holding a number of derivation objects and transforms the derivations' conditions by recursively parsing for nested boolean and comparison operators, nested functions, occurrences of the domain and range variables and literals. The rule parser returns the number of corresponding abstract rule-syntax trees, which matches the number of provided derivations.

RuleInvocation This class requires a completely processed rule container in order to execute the rules in abstract syntax. It recursively applies the parsed operators and calls the stated *reasoning functions* with their parameters. Here the occurrence of a domain or range variable is replaced by the “SpatextBaseData” history of the corresponding entity. If this explicit spatial context is required, the rule invocation accesses one or more federation services via *SpatextBaseDataQuery* objects to retrieve it.

ReasoningFunctions represent an extensible set of functions, which are used for deducing spatial context. They are invoked by the “RuleInvocation” class. In order to increase the performance for rule-based reasoning the functions of this class are hashed when the service is started up.

SpatextBaseDataQuery enables to call the query interface methods of one of the available federation services (see chapter 10.1.1 in the appendix).

6.3.2. Implementation

This chapter explains additional important issues with respect to the actual implementation.

Programming Language The reasoning service was implemented in Java, because in one aspect it is the only language, which is supported by the Jena2 Semantic Web framework. In another aspect Java's introspection capabilities are required for invoking the reasoning functions with adequate parameters.

“Java is high performance. By high performance we mean adequate. By adequate we mean slow.” [EGR99].

Due to the fact that Java is not the fastest language[1] (even small objects are only allocated on the heap, numerous dynamic casts, etc.), everything, that could be cached in main memory, was implemented this way. Therefore all ontologies, the parsed rule containers, that have been hashed, and the reasoning functions can be accessed instantly. This increases the overall performance noticeably.

Of course Java also provides a lot of advantages compared with other object-oriented programming languages such as C++, otherwise it would not have been chosen for the implementation of the reasoning service. Its major benefit is the facilitated software portability to other operating system platforms due to the wide spreading of the Java Virtual Machine. It also provides an automatic garbage collection, thus the programmer is released of the sometimes complicated explicit memory management. To state only some additional nice properties, Java allows for a simplified support for high robustness and safety, and provides lots of useful features already in its core API.

Extended Logging Several (reusable) information such as the queries, ontological declarations, the rule definitions and the reasoning functions have to be provided by SCORE's users – the application developers and system architects. As a matter of fact in addition to system-related exceptions also all possible user errors (misspelled queries, wrong appliance of the SRL, etc.) are printed out with detailed information about the source of the error. This certainly increases the usability of the framework.

Developers who want to modify or extend the reasoning service should use the existing exception-handling classes, which are specialized in managing exception thrown for the query and rule parser, rule invocation and DL reasoner classes.

6.3.3. State of Implementation

- The subscription and event sender functionality is yet not implemented. However this can be added to the service if required. The “SpatextQuery” interface has to be extended with the additional methods for subscription and unsubscription, and a component must be developed, which periodically calls the controllers “handleQuery” method and sends the results to the subscribed applications.
- The DL reasoner does not consider the inheritance of OWL object properties. If the sub-property statements are actually required by applications, it could be added to the reasoner's functionality.
- There is no disintegration of contextual inconsistencies and ambiguities. This would also be the task of the DL reasoner, which would have to be extended in a future project. Note that this does not refer to naming ambiguities, which can be prevented by providing fully qualified resource addresses in the “SpatextBaseData” events (fed-

1 http://www.jelovic.com/articles/why_java_is_slow.htm

eration service) and in the queries for the reasoning service.

- If modifications are applied to the ontologies or the rule base, then the reasoner does not automatically reload the resources. In the case, that multiple applications are using SCORE, so that the reasoning service can not be restarted after any changes, this functionality must be implemented. This can be done by comparing the modification timestamps of the files in the local file system respectively on a web server to the version of the resources in the cache. If there is any deviation, then the DL reasoner must be recreated.
- Currently there is no persistence provided for the data, which results from the reasoning process. As for the federation service, a persistence mechanism was not a design goal for this service. However, if it is desired by application developers, it can be implemented with the help of the Jena API, which also enables persistence management.

6.3.4. Testing

For testing the reasoning service, each of the important functionality-related classes provide special methods, that can be called from the “SpatextReasoningService” class to check on errors or exceptions after the service has been modified. With respect to the integration tests of the distributed services, the “SpatextBaseDataProvider” has been used in combination with multiple federation services.

7. Using SCORE to build new Applications

SCORE is designed to be easily usable for AR applications, that require to be aware of *spatial context* in the users' augmented environments. This chapter explains the setup of the federation and reasoning services for such a new application by means of seven steps.

1. First the entities have to be found out (individual persons and objects in the real and the virtual world), that must be supported by the application. An unique name must be provided for each of these, and they have to be assigned to common classes (e.g. “person”, “aircraft”, “house”, “tree”). Also the binary relations between the classes must be analyzed, which are required for the application logic.

   For instance the relation “player *kicks* football” is necessary in an AR soccer application, in which real persons are playing with a virtual football. All this information can be found out by analyzing the requirements of the individual application.

2. This knowledge must be integrated into SCORE's terminological knowledge base[1]. In order to retrieve the resource address of the root ontology (this might be changed in the future), either the configuration of the reasoning service can be investigated (see figure 50) or simply the reasoning service can be started. In the latter case, the service fetches all ontologies, and prints out both their addresses and the comments in their headers, which provide detailed information about how to extend the particular ontology. Additional ontologies are declared via the “import” statements in each ontology's header. On the condition, that a modification to the ontologies is allowed, a simple text or XML editor or – for more convenience – an OWL ontology editor (e.g. the SWeDE plugin for the Eclipse IDE) can be used to check, which terminology already exists, and which has to be added. In order to keep the knowledge base clearly arranged, new ontologies may be declared depending on the heterogeneity of the information.

```
<!-- file: SpatextReasoningService.xml -->
<attribute name="OWLBaseURI" value=
  "http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/SpatialThing.owl" />
<attribute name="SRLBaseURI" value=
  "http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/SpatextRuleLanguage.xsd" />
```

Figure 50: Excerpt from the reasoning service's description (“OWLBaseURI” states the address of SCORE's root ontology and “ SRLBaseURI” represents the resource address of the spatial context rule language declaration)

3. In the case, that binary relations between classes have been added, the rules must now be specified, that are applied by the reasoning service. For each new OWL object property a RDFS “isDefinedBy” statement has to be provided in order to enable the rule-based reasoning regarding this relation. The given resource address must point to a new rule instance in the rule base, which is currently imported in the header of the root ontology, and can also be extended by new ontology files[2]. A detailed instruction for specifying the actual set of rules is contained in the rule ontology. For each derivation a “return” statement

1 The terminology modeling with ontologies for SCORE is described in chapter 5.3.3.4.1 starting on page 128.
2 The concept of SCORE's rule base is explained in chapter 5.3.3.4.2.

should be given, which will be returned to our querying application, if the respective condition holds. These statements should be defined a way, so that other applications can reuse the rule set.

Because every rule is dependent on certain reusable functions (“distanceBetween()”, “velocity()”, etc.), these can be referenced by the name of the generically implemented methods in SCORE's function repository.

4. If a required function is still missing in the repository, it must be integrated. This is done by providing the actual implementation of the corresponding method in the Java class “*de.tum.in.dwarf.spatext.function.ReasoningFunctions*”. Note that it is often possible to combine the reasoning functions to obtain the necessary functionality. Also other methods, that are already implemented in Java and serve this purpose can often be adapted and reused.
5. Now it is time to configure existing generic spatial context provider services or build one ore more new ones, which might be specific to the application domain. These services must have the ability to send “SpatextBaseData”[1] events, which are received by the federation services. Each event contains the explicit spatial context of one of the entities, which have been analyzed in the first step, and corresponds to an entity class, that has been modeled in the knowledge base in the second step. If other applications will use the same reasoning services, and if the class names might be ambiguous, then the fully qualified OWL class resource addresses should be provided here.
6. In this step the application is developed. It queries the reasoning service via the “SpatextQuery”[2] interface using the query language, which is described in chapter 5.3.3.2.1 on page 115. Also here it might be necessary to define the classes and the property with fully qualified URIs if their short names raise ambiguities. The query response comes in the form of an “SpatextData”[3] array, which contains the deduced context resulting from the query.
7. Before starting up the system, an attribute of the federation service has to be configured first. This is shown in figure 51, and denotes the number of “SpatextBaseData” items, which are cached by the federation service for each entity. In the case, that the attribute is not set, it will be defaulted to “10”.

```
<!-- file: SpatextFederationService.xml -->
<attribute name="maxEntityHistory" value="250"/>
```

Figure 51: Excerpt from the federation service's description

Now the reasoning service can be started. It will detect any errors, which are related to it, and might have occurred when processing the previous steps. In this case it prints out the errors together with additional information, where and why they arose.

Finally all the remaining involved services including one or more instances of the federation and additional reasoning services can be executed.

1 SpatextBaseData is explained on page 107 and declared in chapter 10.2.1 in the appendix.
2 The declaration of the SpatextQuery interface is contained in chapter 10.1.2 in the appendix.
3 SpatextData is described on page 136 and declared in chapter 10.2.2 in the appendix.

8. An Application for SCORE: Spatial Context and Vehicles

During the time for this Master's thesis an application was developed on the basis of SCORE, which is presented in this chapter to show the effectiveness and usability of the framework. Please note that the validation of SCORE's efficiency is topic to the conclusion chapter of this thesis.

This application is integrated into a an ongoing research project for user-centered driver assistance at the Technische Universität München, the Ludwig–Maximilians–Universität München and the university of Regensburg. Since the description of the complete project is clearly beyond the scope of this thesis, I will confine myself to present only those parts of the system, which I developed.

8.1. Introduction

Thinking of vehicles implies thinking of safety above all. This is of special importance in times of more and more rising density of traffic. The increase of passive safety is a fundamental goal when designing new vehicles. However all the approaches such as airbags, active head restraints or the AEB[1] are not aimed at avoiding accidents, but lessen the seriousness of injuries. Certainly there is the requirement for systems like ABS[2], ESP[3] and TCS[4] [CHR02], which are dedicated to actively supervise the safety of the occupants. For this the mentioned systems autonomously interfere during the time of the dangerous driving state.

In the recent years proactive intelligent systems are investigated, which observe the driving situation in order to discover the occurrence of possible threats *before* a dangerous situation arises at all. These systems are able to warn the driver in time for reacting or may also be capable of intervening automatically.

The presented application follows the approach of proactive systems for vehicles by continuously analyzing the state of the own vehicle and surrounding traffic. In the first development iteration the system's task is to provide information in the driving situation of following a vehicle in front and approaching too close, starting to overtake a vehicle in a dangerous situation and being passed by another vehicle while initiating an own overtaking maneuver. These three situations are shown in figure 52.

The information provided by the system is further processed by other applications, which provide it to the driver by using techniques of Augmented Reality.

1 The **A**utomatic **E**mergency **B**reaking is executed if the driver is definitely no more able to avoid a collision [DIT04].
2 **A**nti-lock **B**raking **S**ystem
3 **E**lectronic **S**tability **P**rogram
4 **T**raction **C**ontrol **S**ystem

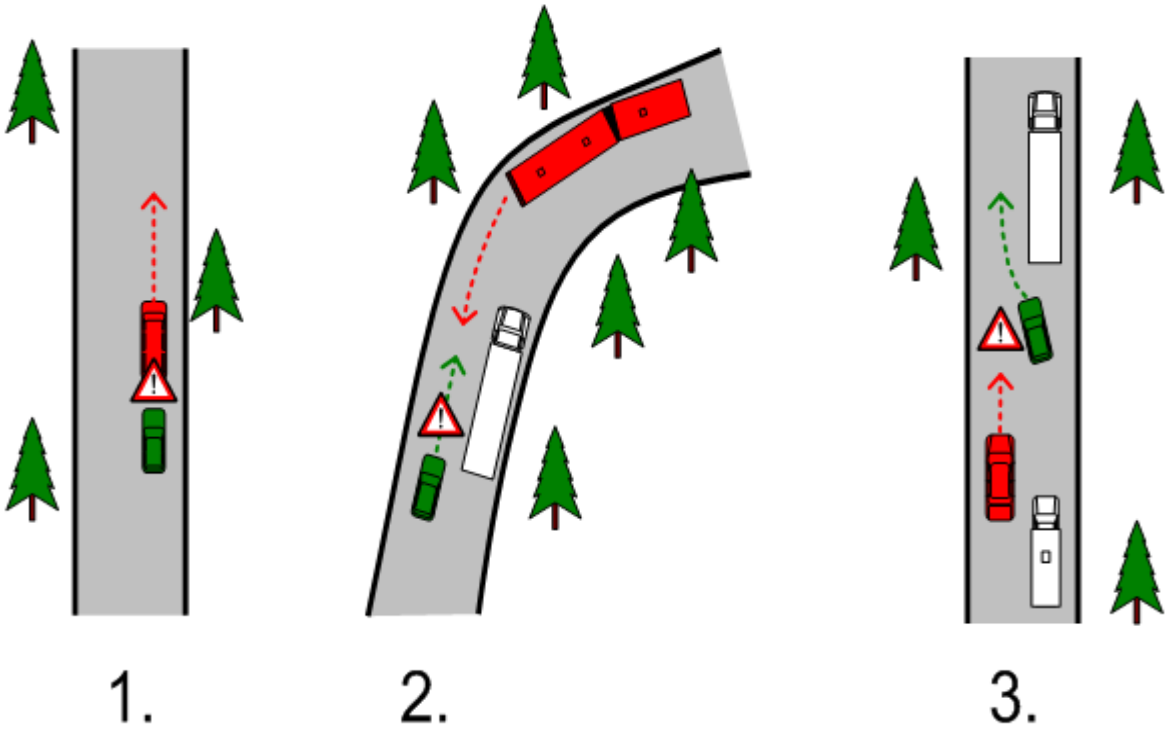


Figure 52: Three different dangerous driving situations regarding the green car: following a vehicle in front (1.), overtaking a vehicle (2.), being overtaken by a vehicle (3.)

Due to the fact, that the development time was not sufficient to provide an implementation for real vehicles, an existing driving simulator is used for demonstrating the application's feasibility. The simulator is shown in the following two figures, that are not further commented. It should only give a first rough idea about the system's environment.

Figure 54: Driving simulator at the faculty of mechanical engineering at the Technische Universität München

Figure 53: A view from the control desk showing the real car and the projected virtual world

8.2. Requirements

In order to support the system's tasks, explicit spatial context of the primary vehicle (i.e. the primary user) must be acquired. This includes the position and orientation in three-dimensional space, the velocity and the acceleration of the vehicle, that is supervised by the application. The context additionally covers the same physical quantities of all secondary vehicles in range. Since the existing driving simulator does not provide an interface to external applications for getting access to the spatial context of the secondary vehicles, its software implementation must be extended for this feature. In this case, the amount of changes to the simulator should be rather small in view of the available development time and the requirement, that other research groups are dependent on it during the development process.

The aggregated information has to be interpreted in order to decide whether or not a situation is dangerous. The latter case implies that another vehicle represents a *spatial obstacle* for the driver. Here it is required to provide *descriptive information*, which must be categorized first, because there is a difference between the driving situations in which the driver has e.g. ten seconds time to avoid a crash or must react instantly.

Due to the fact, that the deduced information is required by other applications in AR, it must be available in “real-time” (see page 14 in the introduction chapter). These applications denote DWARF services, and require the information to be sent via event notification. However also without any connected applications it should be possible to test the functionality of the proactive safety system.

8.3. Design

As mentioned before, the system is responsible for deducing whether or not other vehicles are spatial obstacles for the primary vehicle. Because the information is not directly provided to the driver, but to other applications, that will apply further processing steps, I call the presented system “*Spatial Obstacle Adapter*”. It employs SCORE for context aggregation and reasoning, and behaves as an adapter between it and the supported applications, which in turn need the information about spatial obstacles in a special representation. Since the adapter uses SCORE, its design addresses the steps for creating a new application as described in chapter 7.

Because SCORE currently uses DWARF for dynamically connecting the services, and since the project, for which the adapter represents a subsystem, is also based on this framework, the application is developed as a DWARF service.

8.3.1. Design Goals

The most important design goal is *performance*, because the presented service must have a *low response time* to support other applications in AR. This means, that the *throughput* frequency, in which the complete sequence of information acquisition, aggregation and interpretation is processed, must exceed 16 Hz with respect to the one primary and ten secondary

vehicles. This also includes the whole activity for the communication. Therefore a *low latency time* is required here. The service must also be *reliable* with respect to its specification and *available* to the other applications depending on it with minimal downtimes.

Besides the *scalability* regarding the number of supported applications and the number of vehicles in the simulator, the service must be *modifiable*, if the requirements are changed. Furthermore it must be *extensible* for interpreting additional driving situations and *portable*. The latter criterion in the field of maintenance qualities refers to the *reusable* communication mechanism, which has to be developed for connecting the simulator and DWARF services, that are possibly running on different machines and different operating systems.

Finally the system must be easily *(re)usable*, so that other developers do not have to spent a lot of time in order to get familiar with it.

Compromise Unfortunately I had to decide for a trade-off between the available development time and the system's functionality. Currently the service only provides information for the driving situation, that refers to *following a vehicle* (the left picture in figure 52). However all relevant data for both other cases are already available.

8.3.2. Representing Spatial Obstacle Data

In the context of a traffic situation everything, which is located in the range of the primary vehicle and might collide[1] with it, denotes a *spatial obstacle*. Examples are other vehicles, persons, animals, or static objects such as trees, traffic signs, buildings or also potholes.

Figure 55 shows the categorization of spatial obstacle data by means of its structure in order to enable a common understanding among all involved applications. The data describes information *about* a possible obstacle, which has an effect *for* an entity.

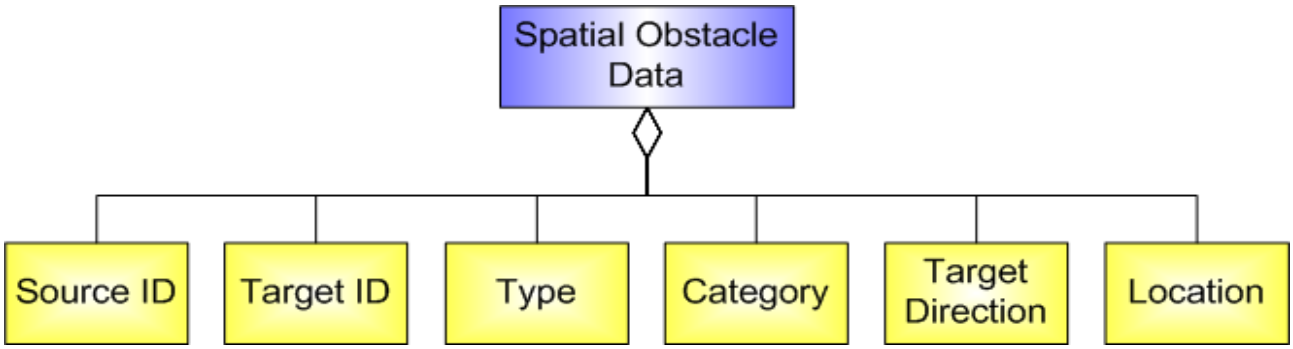


Figure 55: Composition of Spatial Obstacle Data

IDs The source and target IDs represent the identifiers of the primary vehicle and the possible obstacle.

1 Here “collides” is used in the context of software-based collision detections of objects registered in 3D.

Type refers to the obstacle's class. Example types are “unknown_type”, “vehicle”, “building” or “tree”.

Category denotes a finite set of priorities with respect to the obstacle data. It describes the driving state according to the relation between the primary vehicle and the possible obstacle. The category “information” represents a normal, “warning” a potential dangerous and “alert” a clearly dangerous state. An obstacle data with the category “fatal” implies that a collision will definitely occur if the driver is not supported by active safety systems.

The “system_malfunction” category denotes a detected software failure, and “unknown_category” is trivial.

Target Direction tells about the approximate direction, where the obstacle is located on the basis of the primary vehicle's reference point. Examples are “front”, “front_left”, “left”, etc.

Location states the position of the pre-calculated collision, if one is imminent.

Design Rationale Though spatial obstacle data is only directly provided to applications, these will certainly present it to the driver in an appropriate form. Therefore in general it must be clearly understandable by human beings. Its representation contains all important information for further processing and presenting by applications. The IDs can be mapped according to the DWARF Ubitrack[1] namespace or also to SCORE's unique entity names. The *type* allows for distinguishing obstacle classes (the kind of alerting the driver because of a pinhole on the road should differ from the kind with regard to a child jumping in front of the vehicle). The *category* elements are not too many, so that they confuse the driver, but cover all possible conditions. The *target direction* provides sufficient information about the threat's location, which can be computed more precisely via the *location* information of where the collision will take place.

Spatial obstacle data has been explained here by referring to a road traffic situation. However it is also possible to relate it to other domains where the source is a pedestrian, a ship or airplane for instance.

8.3.3. Setting up SCORE

In the analysis of the requirements the entity class “Vehicle” has already been mentioned. An obstacle can be every *spatial thing*, but will be a vehicle in our concrete case. The relation, which describes, that a vehicle is an obstacle for a vehicle (“isSpatialObstacleFor”) is expanded to relate spatial things to *mobile things* in order to be more general. This relation and its definition by means of rule sets is already described as an example in the chapters about

1 http://wwwbruegge.in.tum.de/view/DWARF/ProjectUbitrack

modeling of terminological knowledge (page 128) and SCORE's rule base (page 131). The actual rules can be found in chapter 10.3.3 in the appendix. However one important issue must be explained with respect to the return statements of the spatial obstacle rules. If one of the rules holds, its return statement refers to the *category* and *target direction* of the obstacle. Together with the explicit context of the involved vehicles this information is the foundation for creating the *spatial obstacle data* representation, that will be provided to the applications using the obstacle adapter.

The rules are dependent on the reasoning functions “angleBetween”, “isInFrontOf”, “ttcBetween” (time-to-collision) and “distanceBetween”, which are added to SCORE's function repository during implementation.

The driving simulator or rather a new DWARF service, which mediates between it and SCORE, represents the spatial context provider, and thus will send “SpatextBaseData” events to the federation service. Each event will state information about one of the uniquely numbered vehicles (SCORE's unique ID) of the entity class “Vehicle”. The events of the simulated primary vehicle (its unique ID is “vehicle_0”) must be set accordingly to denote that this entity is the *primary user*. The actual application – the spatial obstacle adapter – uses the following query when accessing the reasoning service.

```
“Vehicle=* isSpatialObstacleFor Vehicle=vehicle_0 as primary user”
```

It will retrieve “SpatextData” describing the implicit relationships between the primary vehicle and all secondary vehicles, that are an obstacle for the driver in the simulator according to the defined rule set. This information will be further processed by the adapter later.

Finally the federation service is configured to keep a history of 100 events per each entity. This is enough in order to enable the reasoning service respectively its reasoning functions to deduce information with the help of past explicit context.

8.3.4. Constituents of the System

The components for an exemplary setup of a running system in the scope of the described application cover the (extended) driving simulator, which is already deployed fixed on a Silicon Graphics ONYX workstation, a socket client, SCORE, the spatial obstacle adapter and a number of client applications. On the next page this setup is shown in figure 56, where the deployment of all components can be changed except for the driving simulator.

Of course any of the client applications, which might run on the windows notebook as shown in the figure, can also directly access SCORE for additional or other information.

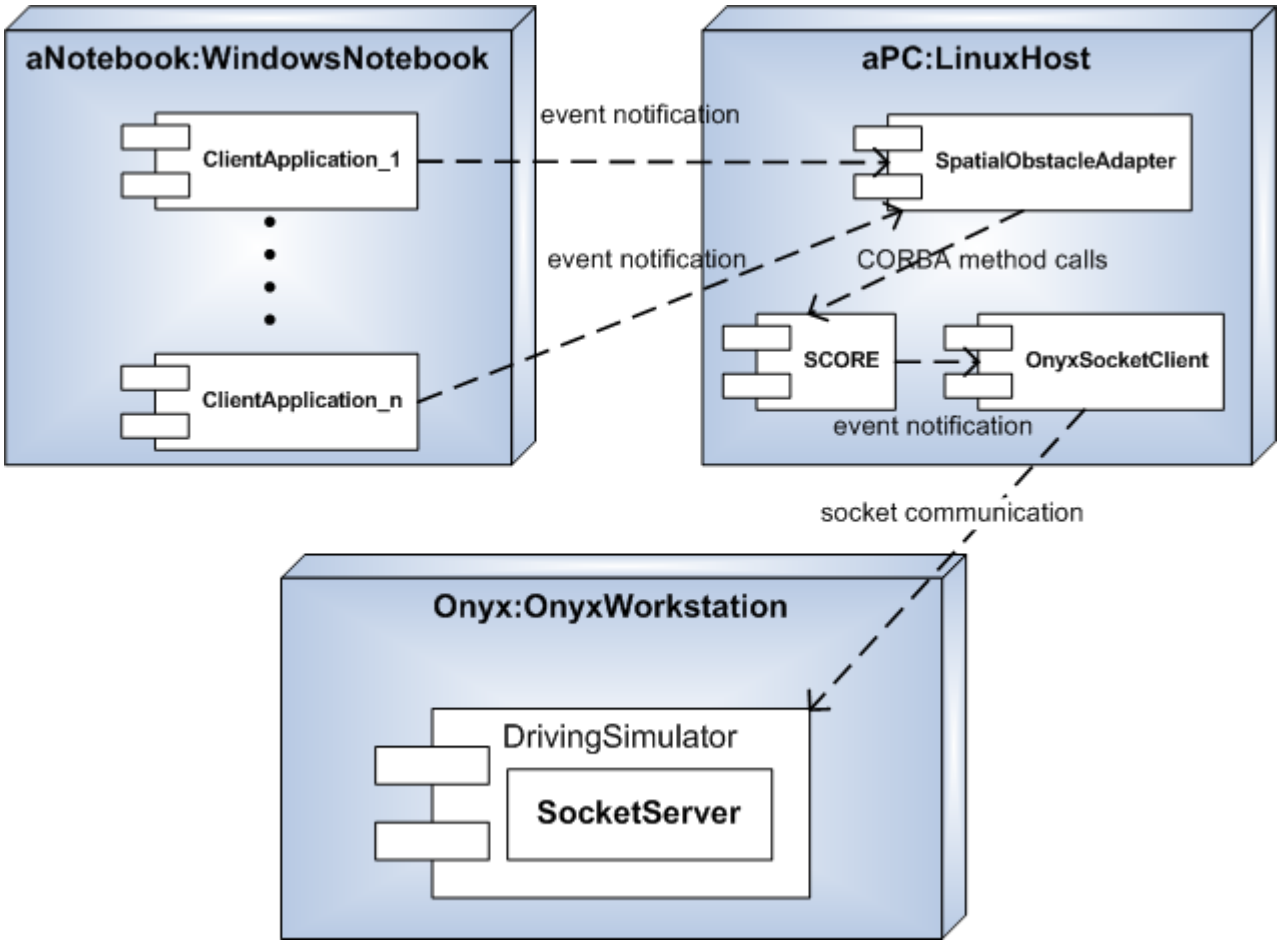


Figure 56: Exemplary deployment of the components in the scope of the "SpatialObstacleAdapter" (UML deployment diagram, dependencies are supplemented by the respective communication mechanism)

8.3.4.1. Socket Communication

The simulator running on the ONYX workstation, does not allow for communicating the spatial information of the simulated vehicles[1]. As a matter of fact it is extended by a socket server, that periodically sends the position, orientation, velocity and acceleration of the primary and all secondary vehicles to the Onyx socket client.

After receiving the data, the “OnyxSocketClient” converts it to a number of “SpatextBaseData” events, which equals the number of the simulated vehicles, assigns them their correct unique names and the identifier “Vehicle” as the entity class, and sends all events to SCORE's federation service (see also figure 57).

Rationale The mentioned communication mechanism between DWARF and the simulator uses sockets instead of the integration of the driving simulator as a DWARF service. The lat-

1 This refers only to the secondary vehicles. Actually the information of the primary car can be listened via a connection over serial ports.

ter case would imply a greater downtime of the simulator for soft- and hardware maintenance, which is not acceptable by other developers and people, who are frequently using it. In contrast its extension by a socket server allows for a rather short development time.

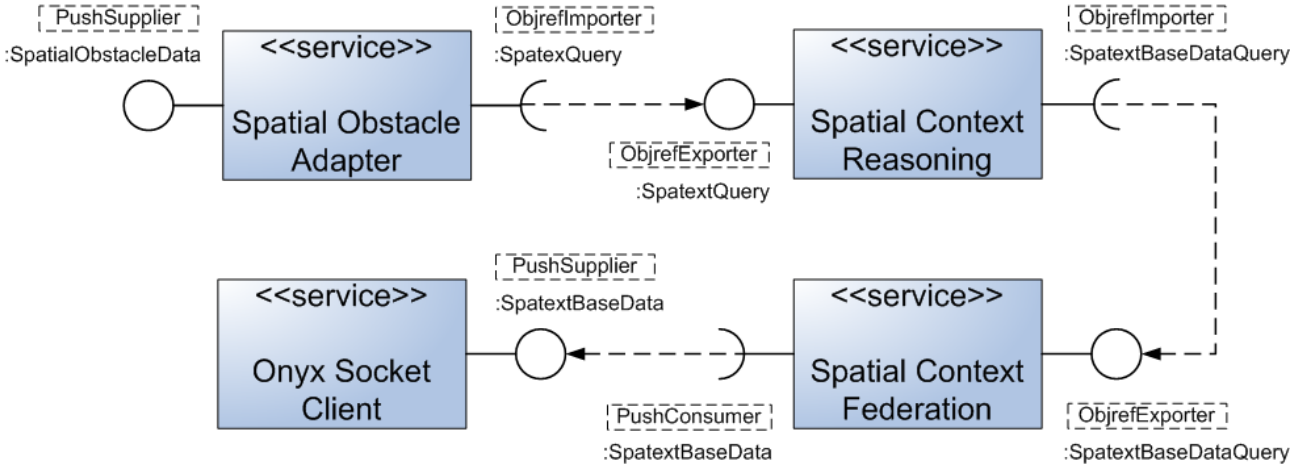


Figure 57: UML diagram showing the connection relationships with respect to the spatial obstacle adapter, SCORE and a socket client

8.3.4.2. SpatialObstacleAdapter

This DWARF service connects to SCORE's reasoner and periodically calls its interface method with the previously presented query (see page 167).

Need For this the service announces a need for accessing the “SpatextQuery” interface. The returned “SpatextData” collection is processed to find the closest obstacle in front with the highest category between the lowest (“information”) and the highest priority (“system malfunction”). This obstacle information is now represented as spatial obstacle data (see chapter 8.3.2) with the primary vehicle as the source reference.

Ability The ability of the service is to send the processed information as “SpatialObstacleData” events (PushSupplier) to client applications. In the case, that no obstacle was detected, the adapter nevertheless sends an event to inform the receivers about the faultless functioning.

Rationale The adapter's current capabilities could also be taken over completely by SCORE[1]. However in view of later extensions to the functionality the decision was made to separate the context interpretation and processing.

1 Apart from the event notification ability, which is not implemented yet.

8.4. Implementation

This section briefly describes the implementation of the spatial obstacle adapter and the socket communication components.

8.4.1. Socket Communication

For both communication sides the *portable* and *reusable* C++ class “SocketBase” was developed, which is compatible with Windows[1] and various Unix derivates as well as Linux. It facilitates the setup of a TCP/IP-based socket connection for clients and servers. For this it provides easy to handle methods for configuring, initializing, using and closing the socket connection. The communication behavior is generalized with the help of the *strategy* pattern, which decouples the sending and receiving of data from the respective application logic.

OnyxSocketClient is implemented as a C++ DWARF service, which is founded on the “SocketBase” class. Because the socket communication to the simulator is synchronized on 20 Hz, the client transforms the received socket data of eleven vehicles to “SpatextBaseData” events in this frequency.

SocketServer Unfortunately it was not possible to extend the C-implementation of the driving simulator with the help of the C++ socket base class. The cause is a missing C++ compiler on the IRIX machine, that could not be installed for lack of available hard disk memory. As a matter of fact, an additional *reusable* pure ANSI C version of the “SocketBase” was implemented and integrated into the simulator, which now sends all vehicles' data to the socket client with 20 cycles per second.

8.4.2. SpatialObstacleAdapter

The adapter is implemented as described in the design. It queries the reasoner and provides the “SpatialObstacleData” events with a frequency of 20 Hz, which is more than analyzed in the requirements.

Testing For testing the functionality the service can be configured to use the “PC speaker” to beep for the obstacle category “information” and the higher ones in different intervals.

1 Via the standard Windows socket library (wsock32.lib).

9. Conclusion

The previous chapters presented a detailed insight of spatial context management for Augmented Reality applications. First an introduction of AR and the DWARF framework was presented, that helps setting up AR applications. After the explanation of the principal aim of this work, the requirements and possible approaches for the management of context in general and spatial context in particular were investigated. Here contextual ontologies were combined with spatial context models to incorporate ontology-based reasoning about common concepts of the augmented world and the scalability of highly dynamic coordinate-based representations of spatial information.

Three frameworks were investigated, that employ different architectures and approaches to handle context on behalf of supported applications. After an overview of helpful technology with respect to widely accepted ontology languages, finally the design of the “Spatial Context Ontology Reasoning Environment” was presented in detail followed by more technical description about its implementation, and the explanation of how easy it is to build new applications with it. Such an application was also demonstrated in the previous section.

Now the point is reached to conclude the thesis. The following section first presents the results, that were obtained by the development of the system followed by its validation and estimations in performance about various kinds of reasoning complexity.

Next some personal experiences are described that I made when working on this thesis, SCORE's design and implementation. The last section of this chapter finally discusses possible future work, that is directly related to this thesis.

9.1. Results

9.1.1. A Framework for Spatial Context Management in AR

The principal result of this thesis is the development of the federation and the reasoning system which together form the SCORE framework.

New Way of Handling Spatial Context The designed framework presents the further evolution of the proposed approach by Becker and Nicklas [BN04], that combines highly dynamic spatial context models and contextual ontologies (see chapter 2.4.3). On this basis and by the separation of terminological and assertional knowledge the system's design involves a whole new form of spatial context management.

As far as all my investigations have revealed, SCORE is the first framework, that is both founded on the new approach and mainly addresses spatial context. However it is yet not possible to state whether it will be accepted by other developers.

AR is not the End The framework was designed in view of providing sufficient performance to enable the real-time requirement of AR. Furthermore it seamlessly integrates the spatial context of physical and virtual entities in the augmented world, that are registered in 3D. Since the system is not dependent on the virtual world, it is also well-suited for any applications, that require only location-awareness or also explicit and implicit spatial context of entities in their domains. This is because SCORE is also able to manage situational location-based information that is only acquired from the real world with the help of additional sensor systems providing spatial information about the users as well as persons and objects in their environment.

Facilitating Query Statements The reasoner's query mechanism allows for abstracting from spatial queries by letting the application developer or system architect define the common spatial relationships between entity classes. Of course it is also possible to just retrieve explicit information for the case, that this is required.

Supporting Application Logic Though in general it is not a good idea to mix framework logic with application logic, SCORE achieves the integration of reusable application logic by enabling a common terminological representation of the application's domain knowledge. The combination of ontological object properties with rules, that can be applied to the dynamic spatial context of any entities, makes it possible to leave the corresponding reasoning tasks to the framework. However SCORE is *not dependent* on the application logic in this case.

Sharing Information By enabling more than only one application to use the same instances of SCORE's services simultaneously, they can share the knowledge managed by the federation and reasoning systems amongst each other.

Another kind of sharing information is related to the common knowledge of application domains. SCORE enables application developers to share their general terminology and defined rules with others, since the information is integrated into a global reusable ontological knowledge base.

A Framework Growing with its Applications Since any new application, that uses the framework will hopefully contribute the one or other reusable reasoning function, new rule sets and shareable ontology information, SCORE's functionality and amount of spatial functions is more and more extended.

A hard to use Framework is useless SCORE is particularly designed to be usable and reusable. The ontologies and the rule mechanisms are easy to understand, above all because they are described by simple examples. Also the query language is very simple, and the rule language orientates itself by the well-known conditional “if” statement of programming languages. The only problematic issue could be the implementation of unavailable reasoning

functions. But due to their usually rather plain structure, this is not too difficult.

Leaving the DWARF Though SCORE currently uses DWARF for connecting the services, the design involves the capability of incorporating it with different systems. For instance that may be Nexus, which could make use of the reasoning system's functionality.

9.1.2. Validation of SCORE

The framework is currently employed in an AR project for user-centered driver assistance. SCORE is already validated by a demonstration of one of the project's parts as described in chapter 8.

Quick development of the Demonstration System This statement is not quite correct, because there are two versions of the same demonstration system. The first was required at the beginning of the thesis, where I have just been analyzing SCORE's requirements. During the somewhat long-winded implementation of the first version I used this chance to gain information for preparing the framework's design.

As a by-product of this implementation also reusable socket communication components were built in order to put up a connection between the simulator and DWARF.

After SCORE was completely implemented, the second version of the demonstration system could then be built rapidly, since only SCORE's setup steps (see chapter 7) had to be gone through.

Successful Demonstration of the Scenario For testing the system, the PC speaker beeps in different intervals to notify the driver about a disregarded safety distance to the vehicle in front, or a rear-end crash. Though this beeping nearly drove me to insanity when trying out different adjustments to the rule set, it did not bother me any more when the system finally worked very well as it is intended to.

9.1.3. Performance

Also the efficiency of SCORE is analyzed. This is done by means of the demonstration scenario, where the sensor data generated by the driving simulator it replaced by spatial information from a number of “SpatextBaseDataProvider” services that can be used to test the performance of SCORE. This was necessary in order to be independent of the simulator. There the number of vehicles can not be changed easily. In contrast the “SpatextBaseDataProvider” allows to configure explicit spatial context of a configurable number of entities.

The concrete scenario is used as the basis of the performance test due to the amount of involved parameters.

The presented performance evaluation is dependent on the following parameters:

- computing resources, number of services and their distribution over the network
- amount of contextual entities, their class memberships, history items per entity and the dynamics of spatial context acquisition
- the used federation service's query interface method and its “history_count” parameter
- the number of different queries to the reasoning system, their kind, complexity and frequency
- complexity of the ontological statements for the application domains
- amount of derivations per query and complexity of involved reasoning functions

Certainly there are also other influencing properties such as the network load, additional simultaneously running applications and so forth.

Setup for the Performance Test The test was carried out using two different computers connected via Ethernet LAN, one federation service, one reasoning service, one “SpatialObstacleAdapter” and a *variable* number of “SpatextBaseDataProvider” services, where each of them itself emulates a number of context providers. Figure 58 shows a DIVE[1] screen shot of one of the test setups, where ten context providers are connected to the federation service.

A DWARF service manager, all “SpatextBaseDataProviders” and the federation service were running on a Linux machine with an AMD XP 1700+ processor and one GB RAM. The reasoning service and the “SpatialObstacleAdapter” were deployed on a Windows notebook using an Intel Pentium M 740 processor and 512 MB RAM.

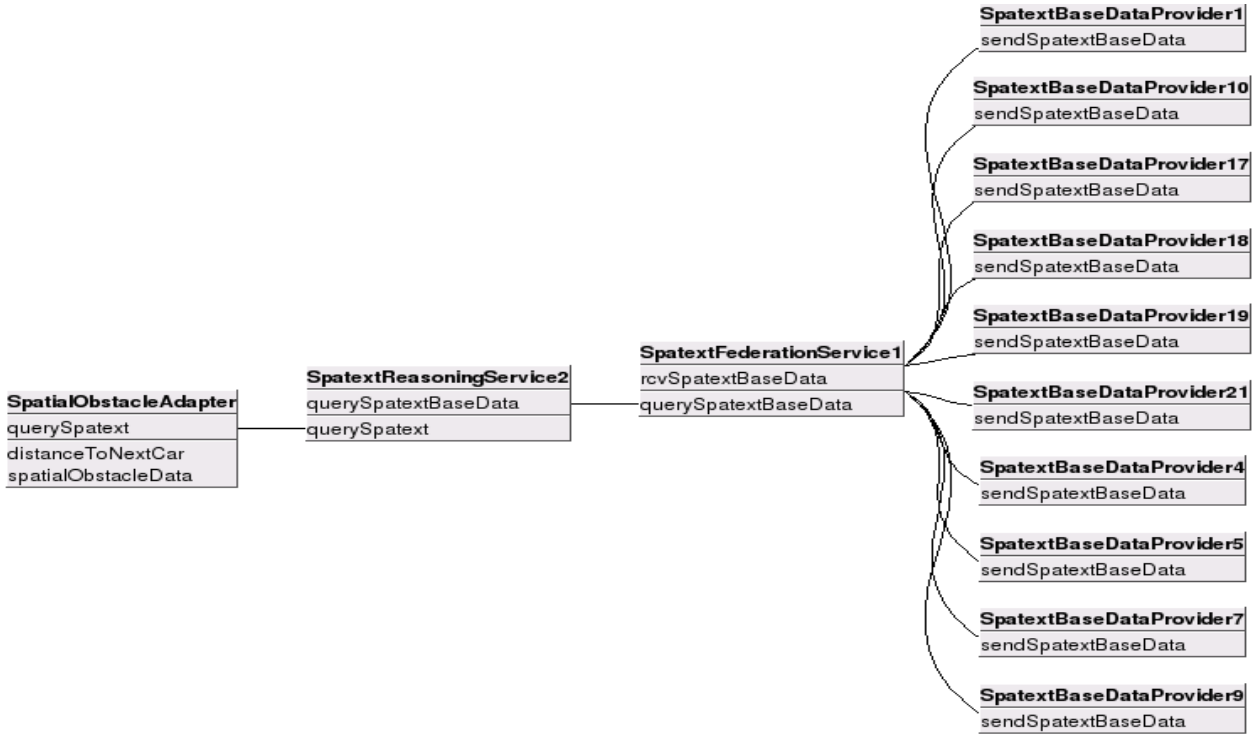


Figure 58: One of the service setups used for SCORE's performance test (screen shot in DIVE, the services are randomly numbered here)

1 **D**WARF **I**nteractive **V**isualization **E**nvironment

Whereas the push frequency of the “SpatextBaseDataProviders” was set to 20 Hz (one event per entity), the obstacle adapter continuously issued the following static n-to-n query.

“*=* isSpatialObstacleFor *=*”

The query above addressed all managed entities, because they were configured as members of the class “MobileThing”, which is the range of this binary relation, and is also subclass of “SpatialThing” representing its domain.

The test addressed a variable number of entities, which was the same for all context providers. Here the federation system was configured to cache 100 history items per entity. This number also refers to the used “history count” when querying the federation service.

With respect to the reasoning service, its initial knowledge base including the “isSpatialObstacleFor” relation and the corresponding rule set was employed (see chapter 10.3 in the appendix). In order to expect a constant reasoning process, the entities' explicit context was predefined a way, so that none of the derivation conditions would hold. Therefore all conditions had to be processed by the reasoner each time a query was issued. The rules reference the functions “angleBetween”, “isInFrontOf”, “ttcBetween” (time-to-collision) and “distanceBetween” in the function repository. Here the first two functions are dependent on past spatial context of the federation service.

This setup was chosen to analyze SCORE's suitability for real applications in dependence of the number of entities and context provider services. Figure 59 shows the overall response times of the federation service's “getEntityHistories” query interface method, that is the most complex one. Querying only the primary user data can be done in constant time, and is therefore not taken into consideration here.

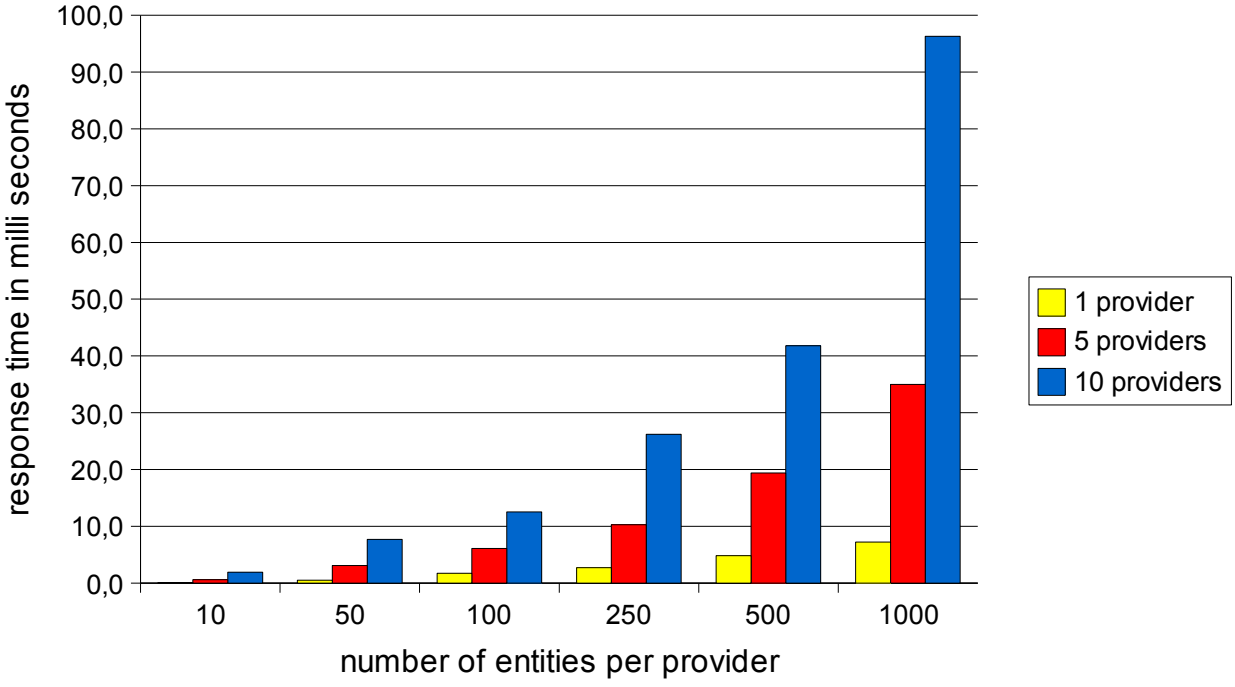


Figure 59: Overall response times of the federation service's “getEntityHistories” interface method (n-to-n query, the most right column is estimated)

As one can see in the diagram, additional providers scarcely effect the performance with respect to their whole entity number compared to one provider, that offers data for the same amount of entities. Please note, that the right blue column corresponding to ten providers with each 1000 entities is an estimation and in contrast to the other values not measured. Here an accurate measurement could not be carried out because 10000 events were too much, so that the context provider services crashed. The estimation is based on the overhead of event communication between the context providers and the federation service.

Bearing the real-time requirement of AR in mind, a federation service is able to manage about 7000 entities while simultaneously processing events (if they arrive) and responding to the reasoning service's requests, that refer to costly n-to-n queries.

If a higher performance must be achieved, then the complexity of the query should be reduced or more federation services can be used that are deployed on different computers. The same consideration relates to the reasoning service, whose query response times are shown in figure 60.

The curve denotes an approximate quadratic increase with respect to the entity number. This corresponds to the quadratic property of the query.

Regarding 100 entities the reasoning service responds to queries in 37,1 milli seconds. Therefore it is suited for applications in AR, that involve approximately this number of entities at most when using queries with quadratic complexity for only one reasoning service.

However if the application is able to distinguish subsets of entities, where a query is stated for each subset, the complexity is reduced noticeably.

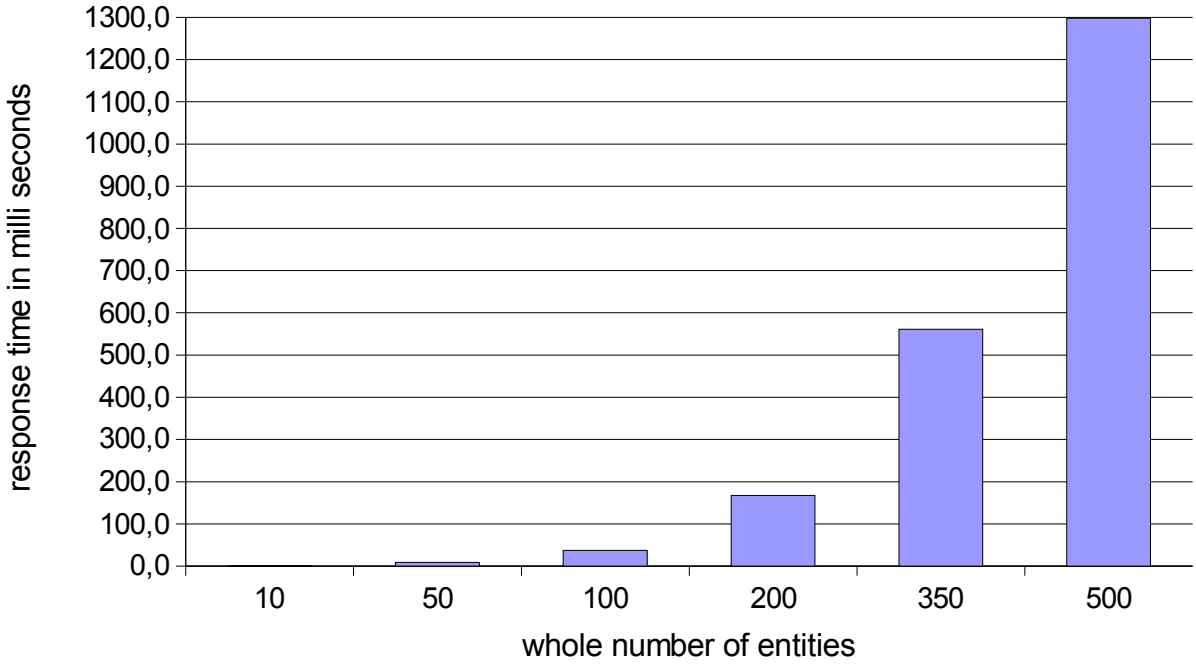


Figure 60: Overall query response times of the reasoning service (n-to-n query)

When dividing up entities based on this approach it must be noted, that a single query will

only span *one* subset of entities. Therefore entities in different subsets are not related to each other with respect to the relation that is stated in the query.

Such an approach is depicted in figure 61 where the 100 entities from the previous performance analysis are split into particular groups of entities so that the number of all entities in all groups equals 100 again. The diagram shows the decreasing number of comparisons while the number of groups is increased.

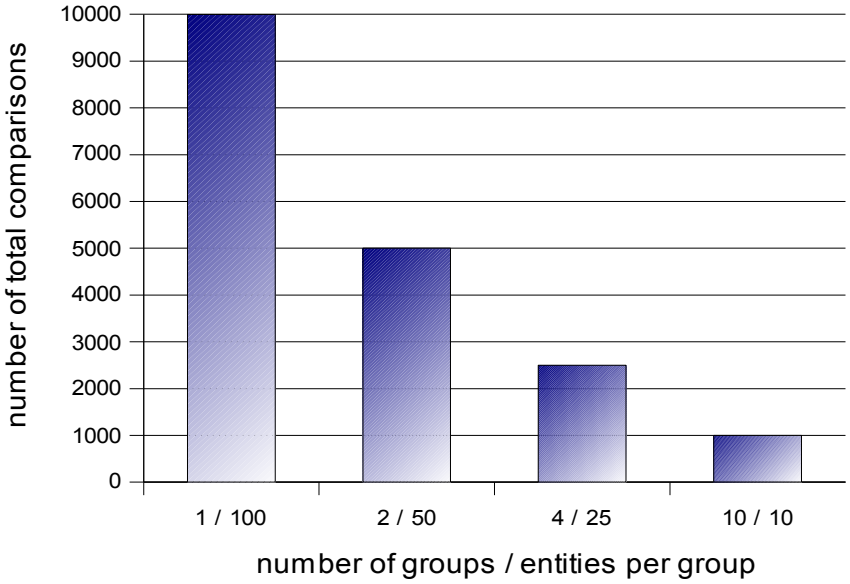


Figure 61: Reducing the complexity of queries by distributing entities over particular numbers of groups

9.2. Personal Experiences

The following topics describe some facts that I experienced during the implementation of SCORE and writing this document. Here the first two issues refer to the C++ programming language.

A little Memory Leak is a big Thing Of course it is, but one will mostly not notice any leaks during C++ coding. They become obvious only when testing, and when one is wondering why the hard disk is working so arduous (due to the relocation of main memory data to it). Though memory leaks can be detected by debugging the system, this is usually rather toilsome. The solution for this problem is “concentrated” programming, i.e. taking notes of any allocated memory on the heap, and freeing it as soon as it is no more required. This is much more efficient than debugging all night long!

Dereferencing invalid Pointers In C++ almost everything is about pointers. Here a good approach for coping with segmentation faults is to always initialize them with the “null”-address. This way it is easy to find out whether a pointer is valid or not. This method should also be followed when deallocating the memory again. Therefore also the problematic with freeing invalid pointers is solved. Note that calling “delete” on a “null”-pointer is harmless.

Writing a Thesis in a foreign Language Writing in English results in the advantage that a lot more people can read it. I hope you did not notice, that I looked up the one or other vocab-

ulary in the dictionary.

9.3. Future Work

In this section I would like to propose some possible topics for a future extension of SCORE's functionality. They also address the points, that were described in the “state of implementation” chapters for both of SCORE's services (6.2.4 and 6.3.3).

Federation Service The implementation of the federation service must be completed. The missing functionality refers to the persistence management of explicit spatial context.

Reasoning Service Here a bit more has to be done. In order to allow for event notification in addition to the method call-based query interface, the subscription component and the event sender must be implemented. Also the DL reasoner should be extended to reason about OWL object property subsumption and to resolve contextual inconsistencies and ambiguities. When the rule base or the ontologies are modified during run-time, the OKB must be reloaded. Finally also a persistence mechanism of implicit spatial context must be integrated.

OWL GUI For a facilitated editing of SCORE's OWL ontologies that describe common knowledge about application domains and rules defining OWL object properties it would be helpful to have a graphical user interface. This GUI should support developers and system architects to work on the reasoner's knowledge base that spans multiple ontologies. Besides editing terminological information the GUI should also provide means for specifying rule instances that are defined using the presented rule language.

Reasoning Functions The function repository contains a rather small set of reasoning functions that were added during the development of SCORE's first application. For enabling developers to use the framework without having to provide implementations for reasoning functions the most important base functions that process explicit spatial context should be determined and realized. An approach for this selection can be the analysis of common spatial relationships between different classes of entities.

For adding the selected functions to the repository already existing related implementations should be investigated and tested so that they can possibly be reused for this purpose.

Security and Privacy Currently the framework does not address security and privacy concerns. However this is very important for the acceptance of the system. These topics should be investigated further on.

Expanding the contextual Scope It would also be very interesting to extend the frame-

work to handle general context. The prerequisites are met by the management of the primary context attributes identity, location and time of entities with respect to particular applications and their users. Now also the reasoning about general and in particular social relationships of entities could be supported.

10. Appendix

The first two sections of the appendix are of special interest to developers of applications and spatial context providers for SCORE. They contain the interface and data type declarations used in the framework. Chapter 10.3 lists the ontologies in OWL and the XML Schema for the “PoseData” data type and the reasoner's rule language.

10.1. SCORE Query Interface Definitions

SCORE provides two different query interfaces, one on the federation layer and one on the reasoning layer.

10.1.1. SpatextBaseDataQuery

This interface provides methods for querying low-level spatial context from the aggregation repository of the Spatext Federation Services.

```
#ifndef __SPATEXTBASEDATAQUERY_IDL
#define __SPATEXTBASEDATAQUERY_IDL

#pragma prefix "in.tum.de"
#include <DWARF/SpatextBaseData.idl>

module DWARF {

     /** @brief Query interface for the spatial context federation service
          some general remarks about this interface:
          -> each method call using the 'unique_identifier' and / or
             'class_identifier' parameter (when it is not 'null') must specify a
             case sensitive string as the corresponding parameter
          -> when using the get*History methods which return a sequence of
             SpatextBaseData then the latest history item of the entity is
             located at the beginning of the sequence (SpatextBaseData[0])
          -> when using the get*History methods which return a matrix of
             SpatextBaseData then the latest history item of each entity is
             located at the beginning of the history sequence
             (e.g. SpatextBaseData[17][0])
     */

     interface SpatextBaseDataQuery{

          SpatextBaseDataSeq getPrimaryUserHistory(in long max_history);

          SpatextBaseDataSeq getNearestEntityHistory(in long max_history);

          /** if 'unique_identifier' is set to 'null' then all aggregated
              entities are returned which belong to class 'class_identifier';

              else if 'class_identifier' is set to 'null' then the entity which
              corresponds to 'unique_identifier' is returned;
```

```
            if 'unique_identifier' and 'class_identifier' are both set 'null'
            then all entities of all classes are returned;

            Additionally specifying the 'class_identifier', the
            'unique_identifier' belongs to (if it is known), increases query
            performance!
        */
        SpatextBaseDataMatrix getEntityHistories(in string unique_identifier,
                    in string class_identifier, in long max_history);

        SpatextBaseDataMatrix getNearbyEntityHistories(in double max_distance,
                    in long max_history);

        /** the following interface methods are special cases of the ones
            above with lHistory = 1 */
        SpatextBaseData getPrimaryUser();

        SpatextBaseData getNearestEntity();

        /** see comment for 'getEntityHistories' which also applies for
            'getEntities' */
        SpatextBaseDataSeq getEntities(in string unique_identifier, in string
                    class_identifier);

        SpatextBaseDataSeq getNearbyEntities(in double max_distance);
    };
};

#endif //__SPATEXTBASEDATAQUERY_IDL
```

10.1.2. SpatextQuery

The following interface allows for querying the reasoning service for higher-level interpreted spatial context.

```
#ifndef __SPATEXTQUERY_IDL
#define __SPATEXTQUERY_IDL

#pragma prefix "in.tum.de"
#include <DWARF/SpatextBaseData.idl>
#include <DWARF/SpatextData.idl>

module DWARF {
    interface SpatextQuery{

        /** returns a sequence of 'SpatextData' items, where each spatext data
            contains the return string of the applied rule, and the spatext
            base data of the domain and range entities, which satify the
            query;
            returns a sequence of length 0 if the query does not apply to any
            entities;
         */
```

```
        SpatextDataSeq query(in string query_string);
    };
};

#endif //__SPATEXTQUERY_IDL
```

10.2. SCORE Data Types for Communication

The listed data types are used with the communication between the SCORE services, applications and the providers of low-level spatial context.

10.2.1. SpatextBaseData

The Spatext Federation Services (SFS) are dependent on CORBA events of type “SpatextBaseData”, which are provided by the context sources. The methods of the query interface, which are implemented by the SFS, also return this data type, sequences of it or matrices of it.

```
#ifndef __SPATEXTBASEDATA_IDL
#define __SPATEXTBASEDATA_IDL

#pragma prefix "in.tum.de"
#include <DWARF/PoseData.idl>

module DWARF {
  struct SpatextBaseData {

     /** unique name of this object, e.g.: "Blue_Car7" */
     string unique_identifier;

     /** name of the class, this object belongs to, e.g.: "Automobile" */
     string class_identifier;

     /** does this object describe spatial basic data of the primary user?
         note: there is exactly one primary user for each application
     */
     boolean isPrimaryUser;

     /** is the velocity field valid? */
     boolean hasVelocity;

     /** is the acceleration field valid? */
     boolean hasAcceleration;

     double velocity;          //in m/s
     double acceleration;      //in m/s^2
     PoseData pose;
  };
```

```
  /** @brief Sequence of SpatextBaseData
   *  @ingroup EventTypes
   */
  typedef sequence<SpatextBaseData>SpatextBaseDataSeq;

  /** @brief matrix of SpatextBaseData items
   *  @ingroup EventTypes
   */
  typedef sequence<SpatextBaseDataSeq>SpatextBaseDataMatrix;

};

#endif //__SPATEXTBASEDATA_IDL
```

10.2.2. SpatextData

This data type is returned by the methods of the query interface, which are implemented by the Spatext Reasoning Service.

```
#ifndef __SPATEXTDATA_IDL
#define __SPATEXTDATA_IDL

#pragma prefix "in.tum.de"
#include <DWARF/SpatextBaseData.idl>

module DWARF {
  struct SpatextData {

     /** the return statement which has been derived from the applied spatial
         context rules by the spatext reasoning subsystem */
     string statement;

     /** a SpatextBaseData container which corresponds to the OWL domain
         instance */
     SpatextBaseData domain;

     /** a SpatextBaseData container which corresponds to the OWL range
         instance */
     SpatextBaseData range;
 };

  /** @brief Sequence of SpatextData
   *  @ingroup EventTypes
   *
   */
  typedef sequence<SpatextData>SpatextDataSeq;
};

#endif //__SPATEXTDATA_IDL
```

10.3. OWL Ontologies and XML Schema Declarations

This appendix lists the initial OWL ontologies “SpatialThing”, “MobileThing” and “RuleBase” together with the complex “PoseData” type and the “Spatial Context RuleLanguage”, which are described in XML Schema. Particularly the ontologies are designed to be reused and / or extended by application developers which use SCORE. The reasoning services are based on this information in order to deduce information about the applications' entities.

Please note, that all the resources, which are directly related to SCORE, are specified in a particular namespace, that was available during SCORE's design. This namespace will be modified to a more appropriate one soon.

10.3.1. SpatialThing (OWL)

This ontology represents extensible common information about entity classes and properties, so that they can be integrated into SCORE.

```
<?xml version='1.0'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY mobthing "http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/MobileThing.owl">
  <!ENTITY rulebase "http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/RuleBase.owl">
  <!ENTITY dts "http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/DataTypes.xsd">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
]>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:dts="&dts;"
   xmlns:xsd="&xsd;"
   xmlns:mobthing="&mobthing;"
   xmlns:rulebase="&rulebase;"
   xml:base="http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/SpatialThing.owl"
>

  <owl:Ontology rdf:about="">
    <owl:versionInfo>$Id: SpatialThing.owl,v 1.5 2005/03/08 16:41:01 fischerj Exp $
    </owl:versionInfo>
    <rdfs:comment>
      An ontology for the Spatial Context Framework created by Jan-Gregor Fischer
      (mailto:fischerj@informatik.tu-muenchen.de).
      Copyright (C) 2000-2005  Technische Universitaet Muenchen
      License: GPL v2 (or later)
      NOTE: the owl:imports statement just tells the resoner to import the resource to the
      ontology model;
      if you link any external resources within this file, e.g. 'rdfs:subClassOf rdf:re
      source="#MobileThing', then you must additionally reference it as a DTD entity (see
      MobileThing.owl)
    </rdfs:comment>
    <owl:imports rdf:resource="&mobthing;#"/>
    <owl:imports rdf:resource="&rulebase;#"/>
  </owl:Ontology>

  <owl:Class rdf:ID="SpatialThing">
    <rdfs:label>SpatialThing</rdfs:label>
  </owl:Class>
```

```
<owl:Class rdf:ID="StaticThing">
  <rdfs:label>StaticThing</rdfs:label>
  <rdfs:subClassOf rdf:resource="#SpatialThing"/>
</owl:Class>

<owl:Class rdf:ID="Building">
  <rdfs:label>Building</rdfs:label>
  <rdfs:subClassOf rdf:resource="#StaticThing"/>
</owl:Class>

<owl:Class rdf:ID="Street">
  <rdfs:label>Street</rdfs:label>
  <rdfs:subClassOf rdf:resource="#StaticThing"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="isSpatialObstacleFor">
  <rdfs:label>isSpatialObstacleFor</rdfs:label>
  <rdfs:isDefinedBy rdf:resource="&rulebase;#SpatialObstacleRule" />
  <rdfs:domain rdf:resource="#SpatialThing"/>
  <rdfs:range rdf:resource="&mobthing;#MobileThing"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="unique_identifier">
  <rdfs:domain rdf:resource="#SpatialThing"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="class_identifier">
  <rdfs:domain rdf:resource="#SpatialThing"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="isPrimaryUser">
  <rdfs:domain rdf:resource="#SpatialThing"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasVelocity">
  <rdfs:domain rdf:resource="#MobileThing"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="velocity">
  <rdfs:comment>the unit of velocity is m/s as defined in SpatextBaseData.idl
  </rdfs:comment>
  <rdfs:domain rdf:resource="#MobileThing"/>
  <rdfs:range rdf:resource="&xsd;double"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasAcceleration">
  <rdfs:domain rdf:resource="#MobileThing"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="acceleration">
  <rdfs:comment>the unit of accelleration is m/s^2 as defined in SpatextBaseData.idl
  </rdfs:comment>
  <rdfs:domain rdf:resource="#MobileThing"/>
  <rdfs:range rdf:resource="&xsd;double"/>
</owl:DatatypeProperty>
```

```
  <owl:DatatypeProperty rdf:ID="pose">
    <rdfs:domain rdf:resource="#SpatialThing"/>
    <rdfs:range rdf:resource="&dts;poseData"/>
  </owl:DatatypeProperty>

</rdf:RDF>
```

10.3.2. MobileThing (OWL)

The “MobileThing” ontology contains the knowledge about the extensible general concepts of mobile entities and their relationships.

```
<?xml version='1.0'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY spt "http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/SpatialThing.owl">
]>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:spt="&spt;"
   xml:base="http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/MobileThing.owl"
>
  <owl:Ontology rdf:about="">
    <owl:versionInfo>$Id: MobileThing.owl,v 1.3 2005/03/08 16:41:01 fischerj Exp $
    </owl:versionInfo>
    <rdfs:comment>
      An ontology for the Spatial Context Framework created by Jan-Gregor Fischer
      (mailto:fischerj@informatik.tu-muenchen.de).
      Copyright (C) 2000-2005  Technische Universitaet Muenchen
      License: GPL v2 (or later)
    </rdfs:comment>
  </owl:Ontology>

  <owl:Class rdf:ID="MobileThing">
    <rdfs:label>MobileThing</rdfs:label>
    <rdfs:subClassOf rdf:resource="&spt;#SpatialThing"/>
  </owl:Class>

  <owl:Class rdf:ID="Vehicle">
    <rdfs:label>Vehicle</rdfs:label>
    <rdfs:subClassOf rdf:resource="#MobileThing"/>
  </owl:Class>

  <owl:Class rdf:ID="Bicyle">
    <rdfs:label>Bicyle</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
  </owl:Class>

  <owl:Class rdf:ID="Automobile">
    <rdfs:label>Automobile</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
    <owl:sameClassAs rdf:resource="#MotorCar"/>
  </owl:Class>

  <owl:Class rdf:ID="MotorCar">
    <rdfs:label>MotorCar</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
```

```
    <owl:sameClassAs rdf:resource="#Automobile"/>
  </owl:Class>

  <owl:Class rdf:ID="Truck">
    <rdfs:label>Truck</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
  </owl:Class>

  <owl:Class rdf:ID="Bus">
    <rdfs:label>Bus</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
  </owl:Class>

  <owl:Class rdf:ID="Person">
    <rdfs:label>Person</rdfs:label>
    <rdfs:subClassOf rdf:resource="#MobileThing"/>
  </owl:Class>

  <owl:Class rdf:ID="User">
    <rdfs:label>User</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>

</rdf:RDF>
```

10.3.3. RuleBase (OWL)

The rule base contains the reasoning rules, which are required by SCORE's reasoning subsystem.

```
<?xml version='1.0'?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:srl="http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/SpatextRuleLanguage.xsd#"
   xmlns="http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/RuleBase.owl"
>
  <owl:Ontology rdf:about="">
    <owl:versionInfo>$Id: RuleBase.owl,v 1.7 2005/03/15 13:52:57 fischerj Exp $
    </owl:versionInfo>
    <rdfs:comment>
      The basic rule ontology for the Spatial Context Framework created by
      Jan-Gregor Fischer (mailto:fischerj@informatik.tu-muenchen.de).
      Copyright (C) 2000-2005  Technische Universitaet Muenchen
      License: GPL v2 (or later)
      Some rules about the spatial context rule language SRL:
        *  boolean operators AND, OR, XOR must be declared in capital letters, and can be
           bound by the '{' and '}' brackets
        *  comparison operators must have exactly two arguments and can be connected by
           boolean operators
           supported comp. operators: &lt;, &lt;=, ==, !=, &gt;=, &gt;
        *  functions can be declared by the function name and the parameter list where the
           latter one is surrounded by '(' and ')' brackets
           each parameter can be a domain or range variable, a function or a literal
           depending on the implementation of this function
           e.g.: ttcBetween(x,y) &gt; 5.5
           note: to define a new function add a static implementation for it to the java
           class de.tum.in.dwarf.spatext.function.ReasoningFunctions
        * access to variable members is possible by prefixing the member as a function,
```

```
          example rule:
          velocity(x) &lt; 33.93
          where x is a domain or range variable
    </rdfs:comment>
  </owl:Ontology>

  <owl:Class rdf:ID="Rule">
    <rdfs:label>Rule</rdfs:label>
  </owl:Class>

  <Rule rdf:ID="SpatialObstacleRule">
    <rdfs:comment>the rule definition for the isSpatialObstacleFor OWL object property
    </rdfs:comment>
    <rdfs:label>SpatialObstacleRule</rdfs:label>
    <srl:variables srl:domain="x" srl:range="y"/>
    <srl:historyCount>100</srl:historyCount>
    <srl:derivations>
      <rdf:Seq>
        <rdf:li>
          <srl:if srl:case="angleBetween(x,y) &lt; 0.3 AND isInFrontOf(x, y) == 1 AND
           {ttcBetween(y,x) &lt;= 10.0 AND ttcBetween(y,x) &gt; 5.5 OR
           distanceBetween(x,y) &lt;=100}">
            <srl:return>information;front</srl:return>
          </srl:if>
        </rdf:li>
        <rdf:li>
          <srl:if srl:case="angleBetween(x,y) &lt; 0.3 AND isInFrontOf(x, y) == 1 AND
           ttcBetween(y,x) &lt;= 5.5 AND ttcBetween(y,x) &gt; 2.0">
            <srl:return>warning;front</srl:return>
          </srl:if>
           </rdf:li>
        <rdf:li>
          <srl:if srl:case="angleBetween(x,y) &lt; 0.3 AND isInFrontOf(x, y) == 1 AND
           ttcBetween(y,x) &lt;= 2.0 AND ttcBetween(y,x) &gt; 1.6">
            <srl:return>alert;front</srl:return>
          </srl:if>
        </rdf:li>
        <rdf:li>
          <srl:if srl:case="angleBetween(x,y) &lt; 0.3 AND isInFrontOf(x, y) == 1 AND
           ttcBetween(y,x) &lt;= 1.6 AND ttcBetween(y,x) &gt;= 0.0">
            <srl:return>fatal;front</srl:return>
          </srl:if>
        </rdf:li>
      </rdf:Seq>
    </srl:derivations>
  </Rule>

</rdf:RDF>
```

10.3.4. PoseData (XML Schema)

The complex pose data type is declared in XML Schema. It encapsulates the simple data types of an entity's pose in the augmented world together with the information about whether or not an entity's position and / or orientation is valid.

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:simpleType name="floatListType">
        <xsd:list itemType="xsd:float"/>
    </xsd:simpleType>
    <xsd:simpleType name="positionType">
        <xsd:restriction base="floatListType">
           <xsd:length value="3"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="orientationType">
        <xsd:restriction base="floatListType">
           <xsd:length value="4"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="poseData" type="poseDataType"/>
    <xsd:complexType name="poseDataType">
        <xsd:attribute name="hasPosition" type="xsd:boolean" use="required"/>
        <xsd:attribute name="hasOrientation" type="xsd:boolean" use="required"/>
        <xsd:attribute name="position" type="positionType" use="optional"/>
        <xsd:attribute name="orientation" type="orientationType" use="optional"/>
    </xsd:complexType>
</xsd:schema>
```

10.3.5. Spatext Rule Language (XML Schema)

The spatial context rule language (SRL) describes the vocabulary and the usage of the rules, which are defined in the rule base ontology.

```
<!-- this language corresponds to de.tum.in.dwarf.spatext.vocabulary.SRL -->
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="
 http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/SpatextRuleLanguage.xsd"
 xmlns="
 http://wwwbruegge.in.tum.de/pub/DWARF/MasterFischer/SpatextRuleLanguage.xsd"
 attributeFormDefault="qualified" elementFormDefault="qualified"
>
  <xsd:element name="variables">
  <xsd:complexType>
      <xsd:attribute name="domain" form="unqualified" type="xsd:string"
        use="required" />
      <xsd:attribute name="range" form="unqualified" type="xsd:string"
        use="required" />
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="historyCount" type="xsd:int" minOccurs="0" maxOccurs="1" />
```

```
  <xsd:element name="derivations">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="if" minOccurs="1" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="return" type="xsd:string" />
            </xsd:sequence>
            <xsd:attribute name="case" form="unqualified" type="xsd:string"
             use="required" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

11. Bibliography

AZU95: Azuma, R. T., "A Survey of Augmented Reality", In Computer Graphics (SIGGRAPH '95 Proceedings, Course Notes #9: Developing Advanced Virtual Reality Applications), pp. 1-38, August 1995. http://citeseer.nj.nec.-com/azuma95survey.html

BCM03: Baader, F., Calvanese, D., McGuiness, D. L., Nardi, D., Patel-Schneider, P. F., "The Description Logic Handbook: Theory, Implementation, Applications", Cambridge University Press, Cambridge, UK, 2003.

BD00: Brügge, B., Dutoit, A. H., "Object-Oriented Software Engineering. Conquering Complex and Changing Systems", Prentice Hall, Upper Saddle River, NJ 07458, pp. 101-105, 2000.

BDG04: Bauer, M., Dürr, F., Geiger, J., Großmann, M., Hönle, N., Joswig, J., Nicklas, D., Schwarz, T., "Information Management and Exchange in the Nexus Platform", Sonderforschungsbereich 627 - Nexus, University of Stuttgart, p. 8, 2004.

BHL01: Beners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, Mai 2001. http://www.sci-am.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam

BN04: Becker, C., Nicklas, D., "Where do spatial context-models end and where do ontologies start? A proposal of a combined approach", chair for Distributed Systems and chair for Applications of Parallel and Distributed Systems, university of Stuttgart, Universitätsstr. 38, 70569 Stuttgart , 2004.

BOR95: Borgida, A., "Description Logics in Data Management", IEEE Transactions on Knowledge and Data Engineering vol.7, No. 5, pp. 472-494, 1995.

BRO96: Brown, P. J., "The Stick-e Document: a Framework for Creating Context-Aware Applications", Electronic Publishing '96, 1996.

CF03: Chen, H., Finin, T., "An Ontology for Context Aware Pervasive Computing Environments", University of Maryland, Baltimore County, Baltimore MD 21250 USA, p. 1, 2003.

CHA02: Chase, N., "XML Primer Plus", SAMS Publishing, 2002.

CHE03a: Chen, H. L., "A Broker-Centric Agent Architecture for Context-Aware Systems", University of Maryland, Baltimore County, Baltimore MD 21250 USA, 2003.

CHE03b: Chen, H. L., "An Intelligent Broker Architecture for Context-Aware Systems", PhD disseration proposal, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 2003.

CHE04: Chen, H. L., "An Intelligent Broker Architecture for Pervasive Context-Aware Systems", dissertation at the Department of Computer Science , University of Maryland Baltimore County, pp. 26, 27, 2004.

CHR02: Christen, C., "Intelligent Safety Systems", Sustainability Report in Economics and Mobility, Adam Opel AG, 2002. http://www.opel.com/corporate/download/opel_intelligent-safety.pdf

CLN98: Calvanese, D., Lenzerini, M., Nardi, D., "Description Logics for Conceptual Data Modeling", Logics for Databases and Information Systems, J. Chomicki and G. Saake eds., Kluwer, pp. 229-263, 1998.

DA99: Dey, A. K., Abowd, G. D., "Towards a Better Understanding of Context and Context-Awareness", 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99), 1999.

DAS99: Dey, A. K., Abowd, G. D., Salber, D., "A Context-Based Infrastructure for Smart Environments", Graphics, Visualization and Usability Center and College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, 1999.

DBF04: Duran-Limon, H. A., Blair, G. S., Friday, A., Sivaharan, T., Wu, M., Okanda, P., Sorensen, C.-F., "Context-Aware Middleware for Pervasive and Mobile Ad Hoc Environments", Department of Computing Science, Tecnológico de Monterrey (ITESM), Campus Guadalajara, México and Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK, 2004.

DEY00: Dey, A. K., "Providing Architectural Support for Building Context-Aware Applications", Georgia Institute of Technology, Atlanta, GA 30332-0280 USA, p. 41, 2000.

DFH02: Davies, J., Fensel, D., van Harmelen, F., "Towards the Semantic Web: Ontology-driven Knowledge Management", Wiley, UK, 2002.

DIT04: Dittmer, M., "Automatic Emergence Breaking", IBEO Automobile Sensor GmbH, System Development, Fahrenkrön 125, 22179 Hamburg, Germany, 2004.

DN00a: Dey, A. K., Newberger, A., "The Context Toolkit: Tutorial: BaseObject", homepage of the Context Toolkit, tutorial section, 2000. http://contexttoolkit.sourceforge.net/documentation/tutorial/BaseObject.html

DN00b: Dey, A. K., Newberger, A., "The Context Toolkit: Tutorial: Context Widgets", homepage of the Context Toolkit, tutorial section, 2000. http://contexttoolkit.sourceforge.net/documentation/tutorial/Widget.html

DSA01: Dey, A., Salber, D., Abowd, G. D., "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", anchor article of a special issue on context-aware computing in the Human-Computer Interaction (HCI) Journal, Volume 16 (2-4), College of Computing & GVU Center, Georgia Institute of Technology and IBM T.J. Watson Research Center, 2001.

EBDN03: Edwards, W. K., Bellotti, V., Dey, A. K., Newman, M. W., "Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Infrastructure", Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto,

CA 94304 USA and Intel Research Lab at Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA 94704 USA, 2003.

EGR99: Egremont, C., "Mr. Bunny's Big Cup O'Java", Addison-Wesley, 1999.

EH91: Egenhofer, M. J., Herring, J., "Categorizing topologial spatial relations between point, line, and area objects"Technical Report, University of Maine, Orono, ME, 1991.

EM92: Egenhofer, M. J., Mark, D. M., "An Evaluation of the 9-Intersection for Region-Line Relations", in: GIS/LIS '92, San Jose, CA, pp. 513-521, 1992.

FEL98: Fellbaum, C., editor, "WordNet: An Electronic Lexical Database", Language, Speech and Communication, The MIT Press, Cambridge, Mass., 1998.

FIS01: Fischer, J.-G., "Introduction of Context-Aware Computing", seminar Methods & Applications for Ubiquitous Computing, 2001.

FK02: Fischer, J.-G., Koch, A., "Sicherheit für Informationssysteme - Replikation, Spiegelung, Verteilung", chair of Database Systems at the Technische Universität München, in German, 2002.

FRA03: Frank, A. U., "Ontology for Spatio-Temporal Databases in: M.e. a. Koubarakis (Ed.), Spatiotemporal Databases: The Chorochronos Approach", Lecture Notes in Computer Science 2520, pp. 9-77, Springer, Berlin, 2003. ftp://ftp.geoinfo.tuwien.ac.at/frank/Chorochronos_chapter2.zip

GAL03: Galloway, A., "Resonances and Everyday Life: Ubiquitous Computing and the City. Draft.", to appear in Cultural Studies, Routledge, under the title "Intimations of Everyday Life: Ubiquitous Computing and the City", 2003. http://www.purselipsquarejaw.org/research_design/papers/galloway_culturalstudies_draft.pdf

GEL01: Gellersen, H.-W., "Ubiquitous Computing", institute for telematics, telecooperation office, Universität Karlsruhe, 2001.

GRU05: Gruber, T., "Introduction to Imaging in Biomedical Informatics", Department of Biomedical Informatics, Columbia University, 2005. http://www.dbmi.columbia.edu/bioimaging/G4032/ontology.htm

HCF97: Hamilton, G., Cattell, R., Fisher, M., "JDBC Database Access with Java: A Tutorial and Annotated Reference", The Java Series, Addison Wesley, 1997.

HM04: Haarslev, V., Möller, R., "RACER User's Guide and Reference Manual Version 1.7.19", Technische Universität Hamburg-Harburg, Inform. and Commun. Dept., Harburger Schloßstraße 20, 21079 Hamburg, Germany, 2004. http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-19.pdf

HNB97: Hull, R., Neaves, P., Bedford-Roberts, J., "Towards Situated Computing"1st International Symposium on Wearable Computers, 1997.

HV99: Henning, M., Vinosky, S., "Advanced CORBA programming with C++", Addison-Wesley Longman, Reading, MA, 1999.

HZ00: Hartley, R., Zisserman, A., "Multiple View Geometry in Computer Vision", Cambridge University Press, 2000.

KBP05: Kato, H., Billinghurst, M., Poupyrev, I., "ARToolkit Documentation", Osaka University, Japan, Human Interface Technology Laboratory at the Universities of Washington and Canterbury, New Zealand, 2005. , http://www.hitl.washington.edu/artoolkit/tutorials.htm#Manual

KE01: Kemper, A., Eickler, A., "Datenbanksysteme", 4. Auflage, Oldenbourg Wissenschaftsverlag, München, Germany, pp. 265-272, 2001.

KOS03: Kost, M., "Web Ontology Language (OWL) bzw. DAML+OIL", in the seminar "Grundlegende Aspekte des Semantic Web", institute for computer science, Humboldt-Universität zu Berlin, in German, 2003. http://www.dbis.informatik.hu-berlin.de/lehre/WS0203/SemWeb/artikel/4/Kost_SemWeb-OWL-Ausarbeitung.pdf

KOS04: Kosub, S., "Effizient Algorithmen und Datenstrukturen", lecture in Computer Science, in German, 2004. http://www4.in.tum.de/~letz/PRUEFUNGEN/kosub-effiziente.pdf

KR01: Kubach, U., Rothermel, K., "Exploiting Location Information for Infostation-Based Hoarding", Proceedings of the 7th Annual ACMSIG-MOBILE International Conference on Mobile Computing and Networking (MobiCom 2001), Rome, Italy, 2001.

LGMR99: Lonley, P. A., Goodchild, M. F., Maguire, D. J., Rhind, D., W., "Geographical Information Systems - Principles, techniques, applications and management", John Wiley & Sons, Inc., New York, 1999.

MAC01: MacWilliams, Asa, "Using Ad Hoc Services for Mobile Augmented Reality Systems, DWARF - Distributed Wearable Augmented Reality Framework", Institute for Informatics at the Technische Universität München, 2001. http://wwwbruegge.in.tum.de/cgi-bin/pub/info.pl?publications/includes/pub/macwilli2001adhoc

MAC05: MacWilliams, A., "A Decentralized Adaptive Architecture for Ubiquitous Augmented Reality Systems", Institute for Informatics at the Technische Universität München, to appear in 2005.

MAN01: Mankoff, J., "Interaction Techniques for Ambiguity Resolution in Recognition-based Interfaces", CoC & GVU Center, Georgia Tech, 2001.

MAT01: Mattern, F., "Ubiquitous Computing - Vision und technische Grundlagen", institute for information systems, ETH Zürich, p. 2, in German, 2001.

MAY99: Mayr, E., "Effiziente Algorithmen und Datenstrukturen", lecture in Computer Science at the Technische Uni-

versität München, 1999. http://wwwmayr.informatik.tu-muenchen.de/skripten/ead_ws9899_html/node21.html
MB95: McGuiness, D. L., Borgida, A., "Explaining Subsumption in Description Logics"In Proc. IJCAI, Montreal, 1995.
MH97: Mahalingam, K., Huhns, M. N., "An Ontology Tool for Distributed Environments", Center for Information Technology, Department of Electrical and Computer Engineering, University of South Carolina, Columbia, SC, USA, 1997.
MV01: Meyberg, K., Vachenauer, P., "Höhere Mathematik 2", Springer Verlag, Berlin, 2001.
OFF00: Offen, R. J., "CASE Tools and Constraints", CSIRO-Macquarie University Joint Research Centre for Advanced Systems Engineering (JRCASE), Macquarie University, North Ryde, NSW 2109, Australia, p. 3, 2000.
OVI02: Oviatt, S., "Breaking the Robustness Barrier: Recent Progress on the Design of Robust Multimodal Systems", Academic Press, vol. 56, 2002.
PRI99: Pritchard, J., "COM and CORBA, Side by Side", Addison Wesley, Reading, MA, 1999.
PS04: Parsia, B., Sirin, E., "Pellet: An OWL DL Reasoner", MINDSWAP Research Group, University of Maryland, College Park, MD, 2004.
PT02: Piekarski, W., Thomas, B., "ARQuake: The Outdoor Augmented Reality Gaming System", ACM Communications, vol. 45, no. 1, pp. 36-38, 2002.
RBB03: Rothermel, K., Bauer, M., Becker, C., "Digitale Weltmodelle - Grundlage kontextbezogener Systeme", Total Vernetzt, Ed. F. Mattern, Springer, 2003.
RBG01: Rolland, J. P., Baillot, Y., Goon, A. A., "A Survey of Tracking Technology for Virtual Environments", in Fundamentals of Wearable Computers and Augmented Reality (chapter 3), pp. 67-112, 2001.
RF00: Rolland, J. P., Fuchs, H., "Optical versus video see-through head-mounted displays in medical visualization", Presence: Teleoperators and Virtual Environments (MIT Press), 2000.
SAW94: Schilit, B. N., Adams, N., Want, R., "Context-Aware Computing Applications", IEEE Network, 1994.
SBWL04: Siu, P. P. L., Belaramani, N., Wang, C. L., Lau, F. C. M., "Context-Aware State Management for Ubiquitous Applications", Department of Computer Science, University of Hong Kong, Pokfulam, Hong Kong, p. 2, 2004.
SCH01: Schöning, U., "Theoretische Informatik - kurzgefasst", Spektrum Akademischer Verlag GmbH, Heidelberg / Berlin, 4. Auflage, pp. 49-51, 80, 91-141, in German, 2001.
SEMD00: Staab, S., Erdmann, M., Mädche, A., Decker, S., "An extensible approach for modeling ontologies in RDF(S)", in: First Workshop on the Semantic Web at the Fourth European Conference on Digital Libraries, Lisbon, Portugal, 2000.
SHGN04: Schwarz, T., Hoenle, N., Grossmann, M., Nicklas, D., "A Library for Managing Spatial Context Using Arbitrary Coordinate Systems", Workshops-Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004), Orlando, Florida, 2004.
SIE99: Siegel, J., "CORBA 3 Fundamentals and Programming", John Wiley & Sons Inc., New York, USA, 1999.
ST94: Schilit, B., Theimer, M., "Disseminating Active Map Information to Mobile Hosts", IEEE Network, 1994.
STR92: Stroustrup, B., "Die C++ Programmiersprache", Addison-Wesley Germany GmbH, chapter 8: Templates, in German, 1992.
SW01: Smith, B., Welty, C., "Ontology: Towards a New Synthesis", Department of Philosophy, University at Buffalo, NY 14260, USA and Computer Science Department, Vassar College Poughtkeepsie, NY 12604 USA, 2001.
TOE03: Tönnis, M., "Data Management for Augmented Reality Applications", 2003. http://wwwbruegge.in.tum.de/cgi-bin/pub/info.pl?publications/includes/pub/toennis2003master
VLI02: van der Vlist, E., "XML Schema - The W3C's Object-Oriented Descriptions for XML", O'Reilly, 2002.
VOL01: Volz, S., "Information Management for Location Aware Applications", in: D. Fritsch & R. Spiller, eds, 'Photogrammetric Week '01', Herbert Wichmann Verlag, Heidelberg, pp. 5–12, 2001.
W3C04a: Heflin, J., Volz, R., Dale, J., "OWL Web Ontology Language Use Cases and Requirements", World Wide Web Consortium, February 2004. http://www.w3.org/TR/webont-req/index.html#onto-def
W3C04b: McGuiness, D. L., van Harmelen, F., "OWL Web Ontology Language Overview", World Wide Web Consortium, February 2004. http://www.w3.org/TR/owl-features/
W3C04c: Heflin, J., Volz, R., Dale, J., "OWL Web Ontology Language Use Cases and Requirements", World Wide Web Consortium, February 2004. http://www.w3.org/TR/webont-req/#section-use-cases
W3C04d: Brickley, D., McBride, B., "RDF Vocabulary Description Language 1.0: RDF Schema", World Wide Web Consortium, February 2004. http://www.w3.org/TR/rdf-schema/index.html
W3C04e: Manola, F., Miller, E., McBride, B., "RDF Primer", World Wide Web Consortium, 2004. http://www.w3.org/TR/rdf-primer/index.html
W3C04f: Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A., "OWL Web Ontology Language Reference", World Wide Web Consortium, February 2004. http://www.w3.org/TR/owl-ref/index.html
WEI91: Weiser, M., "The Computer for the 21st Century", Scientific American, 1991, pp. 94-104.
WHI05: White, S., "Quality Management, in the lecture Software Engineering", University of Houston - ClearLake,

2700 Bay Area Blvd, p. 6, 2005.

WIE04: Wiendl, V., "Robuste Markererkennung für Augmented Reality Anwendungen", 2004. http://mm-werkstatt.informatik.uni-augsburg.de/files/teaching_content/344_RM_AR.pdf

WIE05: Wieland, M., "Ein Framework für Nexus-Anwendungen", Institut für Parallele und Verteilte Systeme, Universität Stuttgart, in German, 2005. http://www.ipvs.uni-stuttgart.de/abteilungen/as/lehre/studentische_arbeiten/diplomarbeiten/DA_Matthias_Wieland/de

ZEI04: Zeidler, A., "A Distributed Publish/Subscribe Notification Service for Pervasive Environments", chair of Computer Science at the Technische Universität Darmstadt, p. 50, 2004.

ZFN02: Zhang, X., Fronz, S., Navab, N., "Visual Marker Detection and Decoding in AR Systems: A Comparative Study", International Symposium on Mixed and Augmented Reality (ISMAR'02), Siemens Corporate Research, Princeton NJ, 2002. http://csdl.computer.org/comp/proceedings/ismar/2002/1781/00/17810097abs.htm