Ein umfassendes Umgebungsmodell als Integrationsstrategie für ortsbezogene Daten und Dienste

Von der Fakultät Informatik, Elektrotechnik und
Informationstechnik der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Daniela Nicklas

aus Temuco (Chile)

Hauptberichter:	Prof. Dr. B. Mitschang
Mitberichter:	Prof. Dr. K. Rothermel
Tag der mündlichen Prüfung:	19. Dezember 2005

Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftware

2005

Ich will dir danken ewiglich, denn du hast es getan.
(Psalm 52,11)

Vorwort

Diese Arbeit entstand an der Abteilung Anwendersoftware des Instituts für Parallele und Verteilte Systeme an der Universität Stuttgart. Themenstellung und Lösung wurden maßgeblich durch meine Mitarbeit in der Forschergruppe Nexus bestimmt, die seit 2003 im Sonderforschungsbereich 627 „Umgebungsmodelle für mobile kontextbezogene Systeme (Nexus)“ fortgeführt wurde.

Mein besonderer Dank gilt Prof. Dr. Bernhard Mitschang, der mir während der gesamten Zeit genau die richtige Mischung aus Freiheit und Anleitung gab, um aus der komplexen Themenstellung unserer Beteiligung am Nexus-Projekt eine Promotionsarbeit heraus zu schälen.

Ebenso möchte ich Prof. Dr. Kurt Rothermel für seine Bereitschaft danken, den Mitbericht zu übernehmen. Als Sprecher des Gesamtprojekts ist er maßgeblich an der Verwirklichung der Nexus-Idee beteiligt. Seine kritischen Anmerkungen und Ideen haben zum Gelingen dieser Arbeit stets hilfreich beigetragen, die ohne sein Engagement nicht in dieser Form stattfinden hätte können.

Bei allen Mitarbeiterinnen und Mitarbeitern des Nexus-Projekts und der Abteilung Anwendersoftware möchte ich mich für die kollegiale und freundschaftliche Zusammenarbeit und Unterstützung bedanken, insbesondere Nicola Hönle, Thomas Schwarz, Matthias Großmann, Martin Bauer, Alexander Leonhardi, Christian Becker, Tobias Drosdol, Frank Dürr, Dominique Dudkowski, Uwe-Philipp Käppeler und natürlich Fritz Hohl für die ursprüngliche Nexus-Idee. Viele der Wissenschaftler im Projekt haben durch ihre Anforderungen, Anregungen und Vorschläge zum Umgebungsmodell beigetragen. Auch die Implementierung der Nexus-Plattform und verschiedener Anwendungen wäre nicht ohne die Zusammenarbeit im Gesamtprojekt und zahlreiche studentische Arbeiten möglich gewesen. Stellvertretend sei an dieser Stelle Jens Messmer, Stjepan Grzan, Michael Moltenbrey, Tobias Frei, Matthias Wieland sowie den Teilnehmern an den Studienprojekten „Spatial Model Server“, „Erweiterter Spatial Model Server“ und der Projektgruppe „Nexus Rallye“ gedankt.

Ohne die Unterstützung meiner Familie und meines Partners wäre die Arbeit sicher nicht zustande gekommen. Herzlichen Dank für Eure Unterstützung, Geduld und Euer Verständnis.

Stuttgart, im Juli 2005 Daniela Nicklas

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AHSS	Aware Home Spatial Server
AKTIS	Amtliches Topographisch-Kartographisches Informationssystem
ASR	Area Service Register (räumlicher Verzeichnisdienst)
AWML	Augmented World Modeling Language
AWQL	Augmented World Query Language
ECS	Extended Class Schema (erweitertes Schema)
HTML	Hypertext Markup Language
HTTP	HyperText Tranfer Protocol
ID	Identifikator
FTP	File Transfer Protocol
GDF	Geographic Data Format
GML	Geographic Markup Language
GPS	Global Positioning System
NSCS	Nexus Standard Class Schema
NOL	Nexus Object Locator
NPL	Navigation Parameter Language
NRL	Navigation Result Language
NSAS	Nexus Standard Attribute Type
NSAT	Nexus Standard Attribute Schema
PDA	Personal Digital Assistant
POI	Point of Interest (Ort von Interesse, z.B. Sehenswürdigkeit)
SCS	Standard Class Schema (Standard-Schema)
SOAP	Simple Object Access Protocol
UML	Uniform Modeling Language
UMTS	Universal Mobile Telecommunications System
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
VIT	Virtual Information Tower (Virtuelle Litfaßsäule)
VTC	Virtual Task Container (Virtueller Aufgaben-Container)
WKT	Well Known Text
WLAN	Wireless Local Area Network
WWW	World Wide Web
XML	eXtensible Markup Language

Zusammenfassung

Der ständige Fortschritt in mobilen Computersystemen und Sensortechnologien ermöglicht eine neue Klasse von Anwendungen: sogenannte kontextbezogene Anwendungen passen sich der Situation (dem Kontext) ihres Benutzers in Präsentation, Informationsselektion und Aktion an. Einen wichtigen Hinweis auf die Situation liefert der Ort eines Benutzers, der über heutige Lokalisierungstechniken verhältnismäßig einfach bestimmt werden kann. Wir sprechen von ortsbezogenen Anwendungen, wenn der räumliche Kontext als primäres Selektionskriterium für die Adaption genutzt wird.

Solche Anwendungen benötigen ein Datenmodell, das Informationen ortsbezogen referenziert, ein sogenanntes Umgebungsmodell. Dieses enthält alle für die Anwendung relevanten Informationen über ihre Umgebung: sowohl Repräsentationen der physischen Welt wie auch digitale Daten und Dokumente, die über das Umgebungsmodell mit der physischen Welt verknüpft werden.

Für eine Vielzahl unterschiedlicher Daten steht durch den Ort ein umfassendes Sortier- und Selektionskriterium zur Verfügung, durch das wir mit der heutigen Informationsflut besser zurecht kommen können – vorausgesetzt, wir können diesen Ortsbezug nutzen. Dadurch wird es möglich, ein umfassendes Umgebungsmodell zu entwickeln, das von verschiedenen ortsbezogenen Anwendungen genutzt wird. Dies hat den Vorteil, dass der hohe Aufwand, Umgebungsmodelle zu erheben und nachzuführen, nicht für jede neue Anwendung erneut auftritt.

Diese Arbeit konzipiert ein solches umfassendes Umgebungsmodell, mit dem ortsbezogene Daten und Dienste zu einer einheitlichen Sicht integriert werden können. Auch für die Verwaltung des Umgebungsmodells wird in dieser Arbeit eine Lösung vorgestellt. Statt einem monolithischen System aus Datenmodell und Anwendung wird eine Föderationsplattform eingeführt, die Datenanbieter und Anwendungen entkoppelt und dynamisch miteinander verbindet. Dies bietet den Anwendungen verschiedene Transparenzen, in Bezug auf das Schema, den Speicherort, die Systemheterogenität und die Verteilung der Umgebungsmodelldaten.

Hierzu wird zunächst eine umfassende Analyse des Anwendungsgebiets ortsbezogener Anwendungen durchgeführt, um inhaltliche Anforderungen an das Umgebungsmodell abzuleiten. Es werden passende Modellierungskonzepte ausgewählt und das Modell aufgebaut. Für die Verwaltung wird eine offene und verteilte Systemplattform entworfen. Die Gesamtlösung wird durch prototypische Implementierung der Plattformkomponenten, Integrationsstudien und die Entwicklung von Beispielanwendungen evaluiert.

Insgesamt wird durch diese Arbeit gezeigt, wie durch Ausnutzung domänenspezifischer Eigenschaften (hier: die räumliche Struktur der Daten) eine effiziente, skalierbare und erweiterbare Föderationsarchitektur bereit gestellt werden kann, die über autonome und wechselnde Anbieter hinweg verschiedenen Anwendungen eine integrierte Datensicht bietet.

Summary

Since powerful computer systems have become mobile, small and affordable, they are our everyday companions: as mobile phones, personal digital assistant, music players or navigation aids. Also, our environment is more and more equipped with electronic devices that not only are able to control everyday things but also to compute, to sense the conditions of their surrounding and communicate with each other. This enables so-called context-aware systems that have knowledge about the situation of their user and adapt their behavior according to that knowledge.

A key challenge in deriving the situation of the user is to analyze her context, e.g. her position in the physical world, near-by objects, the time of the day, her current calendar entry or related web-sites. Getting access to this context is a tedious task for applications: while some of it can be accessed locally (e.g. the user's calendar or the current date), most of it must be obtained from external data sources. It is beneficial to organize the context in so called context models, which can be shared between different applications to reduce the effort of acquisition and maintanance of context data.

This work contributes to the objective of globally shared context model o support various kinds of context-aware applications. Such a model can be realized by a federation platform that uses a common data schema both for representing context information and as an integration strategy for different context sources. The contribution of this schema is the modeling and definition of such a schema, the Augmented World Model, that exploits the spatial organization of context information, i.e. the physical location of objects, to efficiently provide application domain specific access. It connects digital information (e.g. web sites, documents,...) with a model of the physical world that is updated by sensors.

This summary is structured as follows: in the next section the basic requirements of such a platform and a schema for integration are presented. Then we briefly summarize related work argue for the relevance and novelty of the taken approach. An analysis of the application domain gives the requirements regarding the objects of the context model (Augmented World Model). Then, the Nexus platform architecture is pre-

sented, followed by the main features and contents of the Augmented World Model. We evaluate the work by studies of context server integration and application development.

Requirements

The platform that provides the Augmented World Model should fulfill the following requirements:

- access to information based on location, identity and other attributes,
- location-based storage of information (context-aware annotation),
- integration of existing information sources (e.g. the World Wide Web or other context providers),
- deployment of services into the infrastructure to disburden applications on less powerful devices
- extensibility for new applications
- scalability with respect to a high number of data providers and applications
- openness for new providers, new data models, new applications, new services and new information; open and standardized interfaces,
- efficiency; particularly with respect to location- and identity-based access.

The Augmented World Model and its processing concepts should be:

- expressive enough to model the context needed by the applications
- allow for efficient processing and reasoning
- extensible for new context data and schemas
- simple and easy to use

Related Work

There are already a lot of research projects as well as commercial products that develop context- or location-aware applications. However, there is little platform support: most of them manage their own context model that is either built into the application or accessed remotely on specialized servers. The context models used are either not expressive enough for a high variety of applications or too complex to be efficient processable which hampers interoperability and global use.

This holds particulary for ontology-based approaches: typically, they provide a formal and very expressive way of modeling (context) knowledge, but existing reasoning tools and algorithms do not allow for high

amounts of instances (i.e. data objects that are defined by the ontology). Hence, the Augmented World Model is an ontology with a lesser level of interpretation. It also can be shown that no existing geo-data standard can fulfill all requirements for the Augmented World Model.

Other related work covers architectures for information integration, e.g. semantic mediators, GRID computing or enterprise application integration (EAI) systems. These systems have different drawbacks regarding the requirements of this work, e.g. lacking location transparency, not enough dynamism of data providers, inexistence of a global integration schema or inadequate scalability.

Existing platforms for location-based systems are typically not targeted at open and highly dynamic infrastructures and assume that one organization provides the context data for all supported applications (which contradicts the requirement for openness). Additionally, they often do not provide further mechanism for information integration, dynamic data or flexible query interfaces.

Application Domain

To derive the requirements regarding the content of the Augmented World Model (i.e. the kinds of information it has to contain), an analysis of the application domain was made. We can distinguish five major types of context-aware applications:

- Context-aware *information systems* mainly provide documents and other information that is related to the context of the mobile user. Examples are standard location-based services like a hotel/restaurant-finder or location-aware tourist guides.
- *Navigation* applications guide the user from a start point to an end point. Car navigation is already commercial available while advanced mobile pedestrian navigation is still a research topic.
- *Annotation* applications allow for leaving notes or virtual drawings in the environment that can be found by other users. These applications need to update the context model with the new information.
- In *smart environments*, not the user, but her surrounding is equipped with sensors, actuators and devices. Examples are smart meeting rooms that recognize what kind of meeting takes place or the smart factory that bridges enterprise information with the current status of production machines.

- Context-aware *games and edutainment* applications use the physical world as a game board that is augmented with the game context (also known as mixed-reality games). They use fun and competition as incentives for trying out the new technology or learning something about the environment (e.g. a history rally).

These application areas have overlapping requirements regarding the content of their context models. We can derive four major types of needed context information:

- *Geographical context*: digital map data (up to 4D). Physical objects like buildings, streets, rooms; points of interest (POIs) like sightseeing objects, landmarks, etc.
- *Mobile objects*: objects that move, like people, vehicles, or pets.
- *Technical context*: available devices, sensors, communication networks, and infrastructure.
- *Digital information context*: digital information that is relevant to the current place or situation: web sites, documents, notes, or game objects.

The Nexus Platform

The goal of the Nexus platform is to support a large variety of context-aware applications by providing a shared, global context model. To achieve this, the platform federates local models from so called context servers. The local context models typically contain some selection of context information (that seems relevant to the provider) in a spatially restricted area (the service area). Sensors keep local context models up to date.

A *context server* stores a local context model. If one compares the Nexus federation with the WWW, local context servers would correspond to web servers. To participate in the Nexus federation, a context server has to fulfill two requirements: it has to implement a certain interface (simple spatial queries and results in a specified XML language) and it is registered with its service area and object types to the Area Service Register (comparable to a spatially enhanced DNS). There can be many different implementations of a context server, like spatially enhanced databases, legacy systems using a wrapper or even small sensor platforms.

A *federation node* mediates between applications and context servers. It has the same interface as a context server, but does not store context models (except for caching). Instead, it analyses an application query, deter-

mines the context servers that could fulfil the query and distributes the query to these servers. Then it combines the incoming result sets to a consistent view and returns it to the application.

For query distribution and service discovery, a Nexus node uses the *Area Service Register* (ASR). This service is a directory of the available local context models and stores the address of their context server, their object types and the extent of their service areas.

In addition to the query functionality, every Nexus node supports *value-added services*. They use the federated context model to implement advanced services having their own interface (e.g. a map service). These services are also available to clients. Typically, they offer a service that is computational complex and needed by several applications; using this concept, the requirement for deploying services to the platform to disburden less powerful devices can be fulfilled.

A *context-aware application* can use the Nexus platform in three different ways. First, it can send queries to the federation to get context information about its surrounding including infrastructure, points of interest (POIs), mobile objects (friend finder) an so on. Secondly, it can register to a distributed Event Service (not part of this thesis) to get a notification when a certain state of the world occurs, e.g. the user has entered a building or the temperature in a room exceeds a certain value. And thirdly, it can use value added services like the map service to shift basic functions into the infrastructure.

The Augmented World Model

This section describes the main concepts and the contents of the Augmented World Model. Its data model is based on data objects that are formed by attributes. In contrast to object-oriented programming, these data object do not have methods or behaviour. Objects are instances of one or more object types that define which attributes the object must have (*required attributes*) and which attributes it can have (*optional attributes*). An object can have multiple attributes of the same name with different values. The name, structure and basic data type of the attributes is defined in an attribute schema. A class schema imports a attribute schema and groups these attributes to object types. The object types in a class schema form an *is-a* hierarchy (inheritance). If an object type *B*

inherits from an object type *A*, *B* has all attributes of *A* and can add new attributes. Required attributes of *A* must be required in *B*, optional attributes can stay optional in *B* or be defined as required by *B*.

It is possible to extend attribute schemas and class schemas. An extended attribute schema can define new attribute names and structures. As long as it uses the basic data types, the components of the Nexus platform can still process attributes that are compliant to the extended attribute schema. An extended class schema can define new object types as long as they inherit directly or transitively from object types from the base class schema. With this, the Nexus platform can transform objects compliant to the extended class schema to objects of the base schema by omitting the additional attributes (Liskovs substitution principle). This allows applications to use an object of any arbitrary extended type by its type in the base schema. The thesis contains a formalization of this data model and basic operations on it.

The Augmented World Model specifies a so called Standard Class Schema (SCS) that contains object types needed by most of the analyzed context-aware applications. All object types in the SCS inherit from the base class *NexusObject* that defines a required attribute called *type*. This attribute always contains the object type of the object as a string. Hence, the Nexus platform can use this information in processing the context information and applications can query for it. A direct child of *NexusObject* is *NexusDataObject*. Here, a required globally unique ID (the *Nexus Object Locator*), an optional *name* and an optional *description* are defined. Additionally, it has an required attribute named *kind* that either has the value 'real' or 'virtual'. Real objects exist in the physical world, while virtual objects do not: examples are digital information (like web sites or game objects) or planned object (like the model of a not yet existing building). From this object type inherits *SpatialObject,* which is the basis for most of the objects of the Augmented World Model. *SpatialObject* defines the required attributes *pos* (the position) and *extent* (the physical boundary) that contain spatial data in a given coordinate system. For spatial data, the OGC standard GML (*geographic markup language*) is used that allows for different coordinate systems in a geographical location model.

Under *SpatialObject* there are object types defined that model the different types of context derived from the application analysis. The geographical context is given by various object types like *Building, SightseeingObject, RoadElement* and so on. For the mobile context, a base object type *Mobile-*

Object is specialized e.g. by *Person* or *Vehicle*. The digital context is modeled as virtual objects (e.g. a virtual information tower that relates web pages to a spatial region). Finally, the technical context is covered by object types for sensors or communication networks.

For accessing the Augmented World Model, a simple spatial query language named Augmented World Query Language (AWQL) was developed. It allows for selecting object by both standard operators like *equal, lesser, greater, like* and the spatial operators *within* and *overlaps*. The selected objects can be filtered using the *NearestNeighbor* statement (that selects the *k* nearest objects to a given position) and by attribute filters (to select only the attributes the application is interested in). AQWL exploits the object type hierarchy of the Standard Class Schema, and, if given, extended class schemas: if an application queries for objects of a certain type, objects that inherit from this type are also returned. AWQL is defined in an XML-based syntax.

For serializing Augmented World objects, the Augmented World Modeling Language (AWML) was defined. It is also based on XML which allows for flexible marshalling and unmarshalling of the data and the usage of existing tools and parsers. However, the modeling requirements of the Augmented World Model show the limitations of current XML technology: since Augmented World objects can have multiple object types, the type information cannot be given in the meta data (i.e. the element names in the XML document) but we have to use generic objects that contain the type information in the data (i.e. the attribute named *type*). Hence, existing XML parsers cannot validate AWML regarding the class schema but only regarding the attribute schema.

Evaluation

To evaluate the approach of this thesis we implemented prototypes of all platform components as well as different sample applications (with and without platform support) as described in the next section.

To show the extensibility and openness of the architecture, we integrated different existing systems as context servers into our platform:

- The *Aware Home Spatial Server* (AHSS) is a context management platform that was developed for the Georgia Tech Aware Home (a smart environment). It has a simple context model: all relevant objects (like persons, devices,...) are *location objects* that can have arbitrary user-defined attributes. We implemented a wrapper that maps the AHSS

data model to the Augmented World Model and provides AWQL access to the stored context data. Each location is mapped to a *SpatialObject*, or, if the *type* attribute is given, to more specialized object types of the Augmented World Model. AHSS specific data is provided in a extended class schema. Now, existing AHSS applications and Nexus applications can both access the data in the AHSS via different interfaces and can even interoperate if the data schemas match.

- The *Location Service* was developed in the Nexus project especially with the goal of managing the highly dynamic location information for a large number of mobile objects in a scalable way. The scalability of the LS, even for a large number of mobile objects and clients, is achieved through a hierarchical, distributed architecture that contains of several location servers. Each location server is responsible for managing the location information of mobile objects inside a certain service area. If the mobile object leaves this area, the location server performs a hand over to the now responsible server (that is determined by a hierachical distributed index structure).
  This work was prior to the development of AWQL and AWML. Therefore, it does not support a query interface but only functions for position queries or nearest neighbor queries. The data model of the location service was similar to the AHSS.
  We implemented a wrapper that maps AWQL queries to that functional interface that can cope with hand overs of mobile objects between location servers.
- A *Wiki* is a web server that allows any user to edit and insert web pages using a simplified mark-up language. An example is the Wikipedia, an extensive free online encyclopedia to which everybody can contribute. We showed that a wiki engine also can be extended to serve as both a context server and a Nexus application by adding query and insert functions to the Wiki markup and by exporting context objects that are inserted by wiki users to a context server.

We have chosen a variety of different context-aware services and demonstrated their integration in order to assess the suitability of the object model and the query languages with respect to the support of different domains and to prove the extensibility concepts. The experiences gained so far prove the suitability of the concepts of the Nexus platform.

Application Development

The development of mobile context-aware applications differs from traditional software engineering in some aspects: if mobile applications are developed for open environments, they have to cope with more heterogeneity of infrastructures and available data. Also, mobile devices evolve with a high dynamic which necessitates flexible software development processes and architectures. The mobility of the user and the need for context data lead to important design decisions that have major influence of the system model and the targeted application. This thesis states eight major aspects and shows with design decisions have to be taken regarding these aspects.

In the Nexus project, several applications have been developed. In this thesis we analyse four of them on these aspects:

- A simple faculty information system that does not use any external context provider and its later extension with a specialized indoor context server, to show how even simple stand-alone applications can profit from the Nexus platform;
- The *NexusScout*, a mobile city information system that shows several advanced use cases to demonstrate the functional range of the Nexus platform;
- *We are different*, a mobile, location-aware role-play game that shows how the enrichment of already available context information can enable new game ideas and
- *NexusRallye*, a mobile edutainment game that is an example for explorative location-based services; it shows how digital objects (game tasks) can be used to pilot the application operation.

With this experience, a general application design for mobile, context-aware applications can be derived that distinguishes between environmental context (external to the application device), system context (device-internal interfaces and local sensors), service context (external services and other applications) and the user context (for user-centric adaptation and presentation). This thesis mainly covers solutions for obtaining the environmental context.

Conclusion and Future Work

In this thesis we present the Augmented World Model and the Nexus platform, that support various kinds of context-aware applications by flexibly representing context information and integrating different context sources. By restricting the solution to the application domain of context-aware application, we can exploit the specifics of that domain:

- Via an analysis of the application domain we obtained the requirements regarding the concepts and definitions of a Standard Class Schema.
- We can use spatial information as a primary and global index attribute for finding relevant servers by analyzing the queries. This allows for location transparency in the platform.
- The query language can be minimal regarding the requirements of the applications. This enables efficient processing in a distributed, federated information system.

We evaluate the approach by integrating existing information sources and by developing different applications that make use of the platform and the model.

The Augmented World Model and the Nexus platform are a basis for other work in the Nexus project. Hence, its requirements are subject to constant change. Future directions of the approach include:

- Adding higher levels of interpretation to the model and allow for accordant processing in the platform, e.g. the determination of situations that are based on several context information
- Stream-based processing and continuos queries, e.g. for sensor data with high update rates
- Support for inaccurate data and consistency metrics

This work shows how an application-domain specific solution with a global data schema can achieve information maximization, e.g., detection of duplicate objects, merging of multiple representations, generalization or aggregation of data. This is no generic solution and does not solve the problem of information integration. However, other application domains can benefit from our approach by adapting the methods we used: analyse the application domain to build an (extensible) global schema, find global index attributes and define appropriate operators for your query language.

Mut steht am Anfang des Handelns, Glück am Ende.
(Demokrit)

Kapitel 1: Einleitung

Je kleiner und billiger Computersysteme werden (bei gleicher oder steigender Leistung), desto häufiger werden sie zu unserem ständigen Begleiter, ob als tragbarer Rechner, elektronischer Kalender, Mobiltelefon oder tragbares Musikgerät. Auch in unserer Umgebung gibt es mehr und mehr Systeme, die als technische Heinzelmännchen unser Leben erleichtern sollen: von der sich automatisch öffnenden Apothekentür über integrierte Haustechnik bis hin zu sogenannten intelligenten Umgebungen, die sich der aktuellen Situation der dort befindlichen Menschen anpassen.

Ein weiterer Trend ist die zunehmende Vernetzung der Systeme untereinander. Über zahlreiche drahtgebundene und drahtlose Protokolle können unterschiedliche Geräte miteinander kommunizieren und Daten austauschen. Das World Wide Web gibt uns eine immense Menge an verfügbarer Information. Über allgegenwärtige mobile oder in die Umgebung eingebettete Systeme können wir diese Informationen nutzen, ohne an einem Computerarbeitsplatz zu sitzen. Doch je mehr wir Computeranwendungen quasi nebenher einsetzen wollen, desto weniger sind wir bereit, uns von ihnen bestimmen zu lassen. Wir wollen, dass sie uns in der konkreten Situation unterstützen, in der wir uns gerade befinden, ob beim Einkaufen, auf dem Weg zur Arbeit, in der Freizeit oder in einer Besprechung.

Wir sprechen von kontextbezogenen Systemen, wenn sich die Anwendung an den aktuellen Kontext des Benutzers, also seiner Situation, anpasst. Dazu muss die Anwendung den Kontext des Benutzers kennen. Dies ist je nach gewünschtem Detailgrad kein leichtes Unterfangen: sofern die Benutzerin ihren Kontext dem System nicht explizit mitteilt (was in der Regel zu umständlich ist, Wissen der Benutzerin voraussetzt oder besondere technische Kompetenz verlangt), muss das System diesen aus anderen Quellen ermitteln. Dazu verwenden Anwendungen etwa

Daten aus dem Kalender des Benutzers oder bisheriges Verhalten. Für hochaktuelle Kontextinformationen bieten sich Sensoren an, die wie andere elektronische Bauteile ebenfalls immer kleiner und billiger werden und ihre Beobachtungen kommunizieren können. Schwierig bleibt das Problem, aus den unterschiedlichen Kontextinformationen die tatsächliche Situation abzuleiten.

Ein relativ einfach zu bestimmender Kontext ist der Ort des Benutzers. Es existieren bereits zahlreiche Systeme zur Lokalisierung von Personen mit unterschiedlicher Genauigkeit und Verfügbarkeit: das *Global Positioning System (GPS)* ermöglicht es, über Satelliten die Position auf wenige Meter genau zu bestimmen. Voraussetzung ist, dass der Sensor freie Sicht zum Himmel hat. Im Innenbereich können stattdessen infrarot- oder funkbasierte Systeme eingesetzt werden können. Einen Überblick über Lokalisierungssysteme gibt [HB01].

Ortsbezogene Anwendungen passen sich in ihrer Funktion an den Ort der Benutzerin an, in dem sie Informationen nach der Relevanz für einen bestimmten Ort selektieren oder nach der Entfernung zur Benutzerin sortieren. Navigationsanwendungen führen den Benutzer von einem Start- zu einem Zielpunkt. Denkbar sind auch elektronische Geländespiele im Stile einer Schnitzeljagd oder Rallye.

Solche Anwendungen benötigen ein Datenmodell, das Informationen ortsbezogen referenziert, ein sogenanntes *Umgebungsmodell*. Dieses enthält alle für die Anwendung relevanten Informationen über ihre Umgebung: sowohl Informationen über die physische Welt wie auch digitale Informationen, die mit der physischen Welt verknüpft sind.

Frühe ortsbezogene Anwendungen waren isolierte Systeme, d.h. jede verwendete ihr eigenes Modell (z.B. die für eine Navigationsanwendung mitgelieferte CD mit Straßendaten). Diese isolierten Modelle waren zudem meist statisch, d.h. Änderungen der realen Welt (z.B. der Neubau einer Straße) konnten nicht oder nur mit sehr großem Aufwand in den Daten nachgezogen werden (z.B. durch die Auslieferung einer neuen CD).

Die Fortschritte in der Kommunikationstechnik machen es möglich, dass Anwendungen auf Modelle drahtlos zugreifen und diese dynamisch von einer entfernten Quelle abfragen. Damit können die Daten für mehrere Anwendungen von einem Anbieter zentral verwaltet und aktuell gehal-

ten werden. Das Problem hierbei ist, dass ein zentraler Anbieter eine Vorselektion der Daten vornimmt, die u.U. nicht im Sinne der Anwendung und des Benutzers ist. Ein Dienstanbieter, der ortsbasierte Informationen zu Tankstellen zur Verfügung stellt, wird vielleicht nur die Tankstellen der großen Ketten führen, zu denen er auch eine Geschäftsbeziehung hat. Eine nähere, freie Tankstelle würde in seinem Datenbestand fehlen.

Ein Grund für fehlende Umgebungsdaten ist auch, dass es sehr teuer ist, Umgebungsmodelle zu erstellen. Die Daten müssen erfasst, evtl. redaktionell bearbeitet und aktuell gehalten werden. Hierbei kann ausgenutzt werden, dass sich die Modelle verschiedener ortsbezogener Anwendungen inhaltlich und räumlich überlappen. Viele benötigen z.B. Informationen über Straßen oder Gebäude.

Aus dieser Erkenntnis heraus wurde die Idee geboren, ortsbezogene Anwendungen durch ein umfassendes Umgebungsmodell zu unterstützen.

1.1 Die Idee des Umgebungsmodells

Wir leben in einer physischen Welt und sind doch ständig von Informationen umgeben. Viele davon befinden sich an einem Ort, an dem sie relevant sind: Türschilder weisen darauf hin, wer in dem Raum arbeitet, Aushänge sind möglichst so platziert, dass potentielle Interessenten daran vorbei gehen werden, Wegweiser stehen an Weggabelungen und der Abfahrtsplan einer Buslinie findet sich an der dazugehörigen Haltestelle. Auch die früher verbreiteten Ausrufer taten ihre Nachrichten dort kund, wo sie das angestrebte Publikum am ehesten antreffen werden.

Die Vorstellung, dass es zahlreiche Information gibt, die an Orten „lebt“, liegt auch dem Umgebungsmodell zugrunde.

Für die elektronische Fahrplanauskunft ist es sinnvoll, an Bushaltestellen verfügbar zu sein. Die Webseiten von Sehenswürdigkeiten könnten sich in deren unmittelbarer Umgebung aufhalten, so dass Touristen direkt darüber informiert werden. Ein einleuchtender Ort für meine berufliche Webseite ist mein Büro, während meine private sich vielleicht in meiner Wohnung befindet.

Damit sind Informationen über diejenigen Orte zugreifbar, an denen sie von Interesse sind. Eine Benutzerin muss sich nicht unbedingt dorthin bewegen. Sind die Daten erst einmal ortsbasiert zugreifbar, kann sie auch eine virtuelle Reise dorthin unternehmen und sich z.B. schon vor dem Besuch einer Stadt einen Rundgang zusammenstellen.

Für eine Vielzahl unterschiedlicher Daten steht durch den Ort ein umfassendes Sortier- und Selektionskriterium zur Verfügung, durch das wir mit der heutigen Informationsflut besser zurecht kommen können – vorausgesetzt, wir können diesen Ortsbezug nutzen.

1.2 Die Idee der Plattform

Auch für die Verwaltung des Umgebungsmodells soll in dieser Arbeit eine Lösung aufgezeigt werden. Statt einem monolithischen System aus zentraler Datenquelle und Anwendung wird eine Plattform eingeführt, die Datenanbieter und Anwendungen entkoppelt und nur dynamisch miteinander verbindet. Dies hat mehrere Vorteile:

- Anbieter können im laufenden Betrieb hinzukommen. Durch die dynamische Kopplung kann die Plattform diese Anbieter den bestehenden Anwendungen zur Verfügung stellen.
- Anbieter können im laufenden Betrieb wegfallen. Da über die Plattform mehrere Anbieter zur Verfügung stehen, können die Anwendungen weiterlaufen; es fehlen nur die Daten, die der weggefallene Anbieter exklusiv verwaltet hatte.
- Auch Anwendungen können im laufenden Betrieb hinzukommen und wegfallen. Die Anbieter sind davon nicht betroffen, ebenso wenig wie bestehende Anwendungen.
- Die Plattform kann Sensoren von Anwendungen und Anbietern entkoppeln. Damit kann ein Sensor die Modelle verschiedener Anbieter aktuell halten, und die Anwendung wird unabhängig von der konkreten technischen Realisierung einer Realweltbeobachtung.
- Funktionen, die von vielen Anwendungen benötigt werden, können in die Plattform verlagert werden und dort effizient realisiert werden. Dies ermöglicht anwendungsdomänenspezifische Optimierungen und auch die Entlastung leistungsschwacher Endgeräte (wie sie gerade bei mobilen Anwendungen häufig sind).
- Die Plattform kann die verschiedenen Daten der Anbieter zusammenführen und den Anwendungen eine einheitliche Sicht zur Verfügung

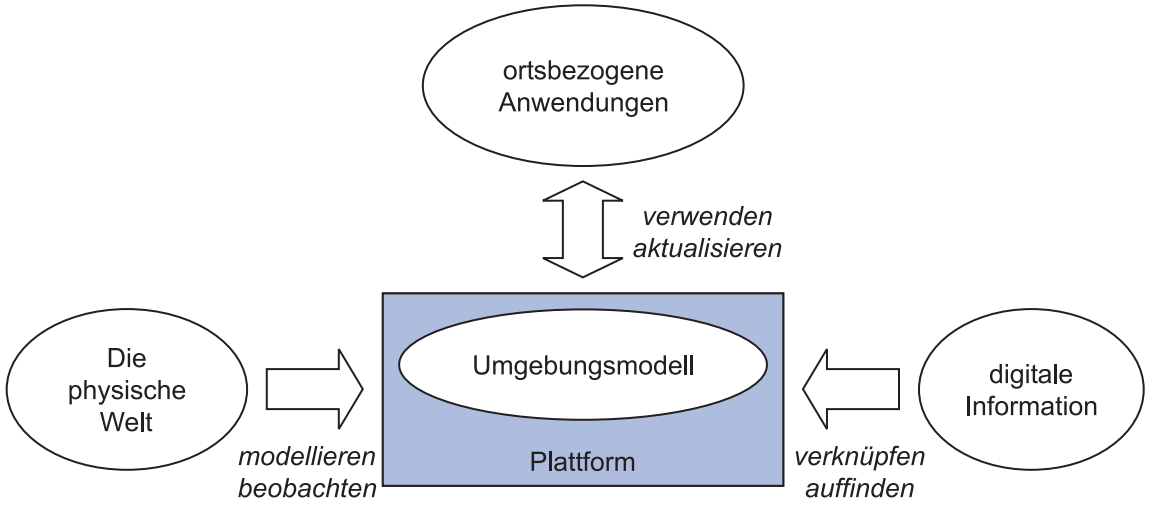

Abbildung 1: Idee dieser Arbeit

stellen. Eine Anwendung muss dann nicht mehr spezifizieren, woher sie Daten benötigt, sondern nur noch, welche Daten sie benötigt. Eine Voraussetzung dafür ist, dass die Plattform Metadaten über die Anbieter kennt, aus denen sie ableiten kann, wie die Datenanfrage einer Anwendung auf die Anbieter abgebildet werden kann.
Die Plattform ermöglicht damit die Verwaltung eines umfassenden Umgebungsmodells, das aus den verschiedenen lokalen Modellen der Anbieter zusammengesetzt wird.

In Abbildung 1 ist die grundlegende Idee dieser Arbeit dargestellt. Um verschiedene ortsbezogene Anwendungen zu unterstützen, soll ein umfassendes Umgebungsmodell entwickelt werden, das Kontextinformationen aus der physischen Welt mit digitalen Informationen verknüpft. Die Aspekte der physischen Welt, die für ortsbasierte Anwendungen relevant sind (z.B. Straßen oder Personen) müssen dazu modelliert werden. Um das Modell aktuell zu halten, müssen Änderungen der Welt beobachtet und nachgezogen werden (z.B. durch Sensoren oder Benutzer, die fehlerhafte Daten melden). Die digitale Information wird mit diesem Modell der physischen Welt verknüpft (z.B. mit Ortsinformation versehen). Unter Umständen muss sie dazu erst aufgefunden werden (z.B. durch Analyse von Webseiten, die Straßenadressen enthalten). Damit Anwendungen das Umgebungsmodell verwenden und ihrerseits aktualisieren können, ist es notwendig, dafür eine Plattform zur Verfügung zu stellen, welche die Daten effizient verwaltet.

Die zentrale Frage dieser Arbeit lautet also:

Wie können mit Hilfe eines Umgebungsmodells mobile, ortsbezogene Anwendungen durch eine offene Plattform unterstützt werden?

1.3 Umgebung dieser Arbeit

Diese Arbeit entstand im Rahmen der Forschergruppe Nexus und dem diese ablösenden Sonderforschungsbereich 627 „Umgebungsmodelle für mobile, kontextbezogene Systeme“ [Nexus]. Während sich die vorliegende Arbeit auf die grundlegenden Themen Umgebungsmodell und prinzipieller Aufbau der Plattform konzentriert, werden im Nexus-Projekt zahlreiche weiterführende Themen bearbeitet, wie z.B. die effiziente Verwaltung mobiler Objekte [Leo03], Datenschutz und Sicherheit [Hau04] oder spezielle Algorithmen zur Generalisierung von Geodaten [HK03], um nur einige zu nennen. Viele der Wissenschaftler im Projekt haben durch ihre Anforderungen, Anregungen und Vorschläge zum Umgebungsmodell beigetragen. Auch die Implementierung der Plattform und verschiedener Anwendungen wäre nicht ohne die Zusammenarbeit im Gesamtprojekt und zahlreiche studentische Arbeiten möglich gewesen.

Der konkrete Beitrag dieser Arbeit ist es, die grundlegenden Entwurfsprinzipien des Umgebungsmodells und der Nexus-Plattform darzustellen, zu begründen, und ein Implementierungskonzept für die Plattform und ihrer zentralen Komponenten aufzuzeigen.

1.4 Lösungsansatz

Für die Entwicklung des Umgebungsmodells und einer Systemplattform wurde folgendes Vorgehen gewählt:

1. Umfassende Analyse des Anwendungsgebiets, um daraus inhaltliche Anforderungen für das Modell zu ermitteln
2. Auswahl passender Modellierungskonzepte und Aufbau eines Modells
3. Konzeption und Entwurf einer offenen Plattform, die das Modell verwalten kann, unter Berücksichtigung von Verteilungs-, Skalierungs- und Erweiterungsaspekten
4. Evaluation der Lösung durch prototypische Implementierung, Integrationsstudien und Anwendungsentwicklung.

Analyse des Anwendungsgebiets. Die zu entwickelnde Lösung soll nicht für sämtliche denkbaren Anwendungen tauglich sein, sondern konkret für das Anwendungsgebiet ortsbezogener Anwendungen. Dies stellt natürlich eine Einschränkung dar, die aber bewusst gewählt wurde: eine spezifische Lösung kann für das Anwendungsgebiet optimiert werden, und die Anforderungen können besser erfüllt werden, wenn die Annahme eines festen Anwendungsgebiets getroffen wird. Im Ausblick (Kapitel 9) wird darauf eingegangen, wie die Erkenntnisse dieser Arbeit und das Systemmodell auf andere Anwendungsgebiete übertragen werden können. Das Anwendungsgebiet ortsbezogener Anwendungen wird in Kapitel 4 analysiert und beschrieben.

Modellierung. Ziel dieser Arbeit ist es, aufzuzeigen, wie ein Umgebungsmodell die Entwicklung einer Plattform für das genannte Anwendungsgebiet unterstützen kann. Dabei müssen folgende Ebenen berücksichtigt werden:

- Daten: Welche Informationen muss das Modell enthalten?
- Datenmodell: Welche Datenobjekte sind modellierbar?
- Meta-Datenmodell: Wie ist das Datenschema aufgebaut?

Einige der Fragestellungen beim Entwurf des Modells können unabhängig von der Systemplattform betrachtet werden. Allgemein gilt jedoch, dass das Modell von der Systemplattform effizient verwaltet werden können soll (Anforderung 9 in Kapitel 2). Manche Entwurfsentscheidungen hängen von der gewählten Systemplattform ab. Deswegen können die Schritte 2 und 3 des Lösungsansatzes nicht sequentiell gesehen werden, sondern müssen überlappend bearbeitet werden. In dieser Arbeit wird das Umgebungsmodell in Kapitel 6, also nach dem Entwurf der Systemplattform, beschrieben.

Systemplattform entwerfen. Der dritte Schritt des Lösungsansatzes besteht im Entwurf einer Systemplattform, die ortsbezogene Anwendungen unterstützt. Dazu müssen folgende Fragestellungen bearbeitet werden:

- Funktionale Komponenten: Welche Aufgaben sollen erfüllt werden?
- Verteilung der Komponenten: Welches ist eine sinnvolle Verteilung für diese Komponenten?
- Datenfluss: Wie sieht der Datenfluss und damit die Kommunikation zwischen den Komponenten aus?

Evaluation. Schließlich soll die Lösung auf ihre Brauchbarkeit hin evaluiert werden. Hierzu wurden verschiedene Integrationsstudien durchgeführt sowie Anwendungen entwickelt, die auf der entwickelten Systemplattform aufsetzen.

1.5 Überblick

Kapitel 2 präzisiert das Thema dieser Arbeit. Durch eine Reihe von Anforderungen wird dargestellt, welches konkrete Problem diese Arbeit lösen möchte. In Kapitel 3 wird erläutert, welche verwandten wissenschaftlichen oder kommerzielle Arbeiten bereits bestehen und warum diese die Problemstellung bestenfalls nur teilweise lösen.

Die folgenden drei Kapitel beschreiben die Lösung an sich: Kapitel 4 definiert die Anwendungsdomäne und konkrete Anwendungsfälle, Kapitel 5 stellt die Architektur der angestrebten Plattform vor und Kapitel 6 beschreibt die zentralen Eigenschaften und Bestandteile des Umgebungsmodells.

Um die Brauchbarkeit der Lösung zu demonstrieren und zu evaluieren, werden in Kapitel 7 Möglichkeiten dargestellt, wie das Modell in verschiedene Richtungen angepasst, erweitert und weiterentwickelt werden kann, während Kapitel 8 bestehende Anwendungen beschreibt, die Plattform und Umgebungsmodell nutzen.

Mit Kapitel 9 schließt die Arbeit.

Requirements are like water. They're easier to build on when they're frozen. (Anonym)

Kapitel 2: Problemstellung

In diesem Kapitel soll die Problemstellung der Arbeit beschrieben werden. Die Arbeit folgt der grundlegenden Fragestellung:

Wie können mit Hilfe eines Umgebungsmodells mobile, ortsbezogene Anwendungen durch eine offene Plattform unterstützt werden?

Im Folgenden werden nun die verschiedenen Bestandteile der Frage erläutert und definiert, um daraus die Anforderungen an die Lösungsansätze abzuleiten. Dazu wird zunächst die Anwendungsklasse beschrieben, dann die Anforderungen der Plattform und schließlich die des Umgebungsmodells.

2.1 Anforderungen der Anwendungsklasse

2.1.1 Ortsbezogene Anwendungen

In der heutigen Gesellschaft spielt Mobilität eine große und immer noch wachsende Bedeutung. Auch Computeranwendungen sind nicht mehr an Großrechenanlagen oder Arbeitsplatzsysteme gebunden, sondern können aufgrund der fortschreitenden Miniaturisierung auf immer kleiner werdenden Endgeräten überall benutzt werden. Mobile Anwendungen zeichnen sich also dadurch aus, dass die Benutzerin mobil ist. Dies hat sowohl Auswirkungen auf die geeignete Präsentation (kleine Bildschirme, verminderte Aufmerksamkeit) als auch auf die Funktionalität: unterwegs sind andere Anwendungen wünschenswert. Besonders interessant ist es, die sich ändernde Situation der Benutzerin in der Anwendung zu berücksichtigen. Allgemein spricht man hierbei vom Kontext

der Anwendung (oder der Benutzerin), wobei der Ort häufig eine wichtige Rolle spielt. Ortsbezogene Anwendungen sind also ein Spezialfall kontextbezogener Anwendungen.

Es existieren zahlreiche Definitionen von Kontext (z.B. [DA99], [CK00]). In dieser Arbeit soll die Definition von [RBB03] verwendet werden:

> **Definition 1: Kontext**
> *Kontext ist die Information, die zur Charakterisierung der Situation einer Entität herangezogen werden kann. Entitäten sind Personen, Orte oder Objekte, welche für das Verhalten von Anwendungen als relevant erachtet werden. Dabei wird eine Entität selbst als Teil ihres Kontexts betrachtet.*

Darauf aufbauend wird eine kontextbezogene Anwendung wie folgt definiert:

> **Definition 2: Kontextbezogene Anwendung**
> *Eine Anwendung ist kontextbezogen, wenn ihr Verhalten durch Information über ihren Kontext beeinflusst wird.*

In [RBB03] wird zwischen Primärkontext (Ort, Identität, Zeit) und Sekundärkontext (daraus abgeleiteter Kontext, z.B. Zustand oder Aktivität) unterschieden. Ortsbezogene Anwendungen sind also solche, die hauptsächlich über den Primärkontext Ort auf Informationen zugreifen. Diese Information wird auch als *Umgebungsinformation* bezeichnet.

Im Folgenden werden nun zentrale Anforderungen beschrieben, die sich aus den Eigenschaften mobiler, ortsbezogener Anwendungen ergeben. Dabei werden zunächst die Anforderungen beschrieben, die Anwendungen an eine sie unterstützende Plattform stellen (für die Idee der Plattform siehe auch Kapitel 1.2), dann Anforderungen an den Entwurf der Plattform (Kapitel 2.2) und schließlich Anforderungen an das Umgebungsmodell (Kapitel 2.3).

2.1.2 Anforderungen

> **Anforderung 1: Ortsbasierter Zugriff auf Information**
> *Anwendungen sollen einen ortsbasierten Zugriff auf Informationen erhalten. Dies bedeutet, dass Anwendungen Anfragen mit räumlichen Prädikaten stellen können, auf die sie Ergebnisse über Informationen in*

ihren Ortskontext erwarten, wie z.B. "Wo befindet sich das nächste italienische Restaurant?"

Anwendungen greifen nicht nur passiv auf Umgebungsinformation zu, sondern verändern diese auch selbst. So könnte beispielsweise in einem elektronischen, kollaborativen Reiseführer Reisende ihren Eindruck zu Sehenswürdigkeiten direkt vor Ort hinterlassen:

Anforderung 2: Ortsbasiertes Ablegen von Information
Anwendungen sollen über die Plattform Informationen ortsbasiert ablegen können. Dies schließt sowohl das Ändern vorhandener Information wie auch das Anlegen neuer Information mit ein.

Auf Umgebungsinformation ausschließlich ortsbezogen zuzugreifen, reicht jedoch nicht aus. Eine Anwendung könnte der Benutzerin zunächst grobe Informationen über ihre Umgebung liefern (eine Liste von Restaurants). Die Benutzerin wählt darauf hin ein Restaurant aus, zu dem sie nähere Informationen möchte. Die Anwendung sollte nun in der Lage sein, über eine eindeutige Identifikation (ID) Zusatzinformationen über das Restaurant abzurufen. Auch für den schnellen Abgleich von Objekten ist es wichtig, dass diese eine eindeutige ID haben:

Anforderung 3: ID-basierter Zugriff auf Information
Anwendungen sollen auf Informationen über eine eindeutige ID zugreifen können. Diese ID soll Teil des Resultats sein, das die Anwendung auf einen ortsbasierten Zugriff zurück enthält. Damit soll es möglich sein, zu derart identifizierten Entitäten weitere Informationen zu erhalten.

Ortsbezogene Anwendungen entstehen nicht im leeren Raum, sondern sie sind eine Ergänzung zu bestehenden Informationsdiensten, wie z.B. dem Web. Die Plattform sollte nicht erfordern, dass diese Technologie neu erfunden oder dupliziert wird, sondern Brücken in bestehende Systeme bauen:

Anforderung 4: Integration bestehender Informationsdienste
Bestehende Informationsdienste, wie z.B. das Web, sollen über die Plattform integriert werden. Informationen aus externen Quellen sollen über ortsbasierte Zugriffe über die Plattform von Anwendungen abgerufen werden können.

Endgeräte für mobile Anwendungen sind trotz rasanter technologischer Entwicklung meist wesentlich weniger leistungsfähig als stationäre Systeme. Mobile, ortsbezogene Anwendungen müssen deswegen mit Hardware-Ressourcen sparsam umgehen. Die Plattform sollte deswegen in der Lage sein, rechenintensive Operationen infrastrukturbasiert auszuführen:

Anforderung 5: Verlagerung rechenintensiver Operationen in die Plattform
Die Plattform soll in der Lage sein, rechenintensive Operationen der Anwendung auszuführen, um schwache Endgeräte zu entlasten, wie z.B. das Zeichnen von dynamischen Karten aus der Umgebung oder die Berechnung einer Navigationsroute.

Obwohl es heutzutage schon Beispiele für ortsbezogene Anwendungen gibt, sind sie immer noch ein Wachstumsmarkt und Forschungsgegenstand. Es gibt immer wieder neue Anwendungsideen, die umgesetzt werden sollen. Es ist deswegen nicht möglich, einzelne Anwendungen fest zu definieren, auf welche die Plattform optimiert wird, sondern die Plattform soll offen sein für Neuentwicklungen:

Anforderung 6: Erweiterbarkeit bezüglich neuer Anwendungen
Es soll einfach möglich sein, neue Anwendungen an die Plattform anzubinden. Dies erfordert Flexibilität bezüglich möglicher Anfragen, des Datenschemas und auch bezüglich der Operatoren, die innerhalb der Plattform ablaufen (siehe Anforderung 5).

2.2 Anforderungen an den Plattformentwurf

In diesem Abschnitt werden nun Anforderungen aufgeführt, die sich direkt an den Entwurf der Plattform selbst richten.

Anforderung 7: Skalierbarkeit
Die Plattform soll für den globalen Einsatz geeignet sein. Dies bedeutet eine Vielzahl verschiedener Anwendungen und Anwendungsinstanzen und eine Vielzahl von Datenanbietern.

Anforderung 8: Offenheit
Die Plattform soll bezüglich mehrerer Seiten offen sein:

- *neue Datenanbieter: es soll einfach möglich sein, neue Datenanbieter an die Plattform anzuschließen, so dass bestehende Anwendungen ohne Einbußen weiterlaufen können.*
- *neue Anwendungen: es soll einfach möglich sein, dass neue Anwendungen hinzukommen, ohne dass bestehende Anwendungen geändert werden müssen.*
- *neue Dienste: es soll einfach möglich sein, dass weitere Dienste zum Funktionsumfang der Plattform hinzukommen.*
- *neue Informationen: es soll einfach möglich sein, neue Informationen in die Plattform einzubringen, auch wenn ihre Struktur bisher nicht vorgesehen war (Schemaerweiterung).*

Diese Offenheit soll insbesondere erleichtert werden durch:

- *offene Schnittstellen: die Schnittstellen der Plattform sollen offengelegt werden.*
- *offener Quellcode: der Quellcode der Plattform soll offengelegt werden.*

Es wird sich später zeigen, dass die Offenheit eine zentrale Anforderung ist, die an vielen Stellen zu wichtigen Design-Entscheidungen führt. Es ist ein großer Unterschied, ob ein System für den geschlossenen Einsatz entwickelt wird, bei der zentrale Institutionen, z.B. Systemadminstratoren die Kontrolle besitzen, oder für einen offenen Einsatz, bei dem zahlreiche verschiedene Betreiber lose miteinander kooperieren, um einen gemeinschaftlichen Nutzen zu erlangen.

Definition 3: Lokalitätsannahme
Die Lokalitätsannahme besagt, dass ortsbezogene Anwendungen häufiger Information aus ihrer eigenen Umgebung anfragen als solche, die weit entfernt liegen.

Anforderung 9: Effizienz
Die Plattform soll für die geplanten Anwendungen effizient arbeiten. Diese Anforderung bezieht sich insbesondere auf den ortsbasierten und den ID-basierten Zugriff auf Information unter der Lokalitätsannahme.

Natürlich wäre ein System wünschenswert, das unter sämtlichen Randbedingungen effizient arbeitet. Dies ist aber selten möglich. Deswegen müssen Kompromisse geschlossen werden. Die Lokalitätsannahme ermöglichst es dabei, auf bestimmte und als realistisch angenommene Szenarien hin zu optimieren.

2.3 Anforderungen an das Umgebungsmodell

Bevor die wesentlichen Anforderungen an das Umgebungsmodell skizziert werden, wird zunächst der Begriff „Umgebungsmodell“ definiert:

> **Definition 4: Umgebungsmodell**
> *Ein Umgebungsmodell besteht aus Daten, welche die Umgebung eines Benutzers abbilden. Zusätzlich kann es auch Referenzen oder Metaphern für digitale Informationen enthalten, die für die Umgebung des Benutzers relevant sind. Der Benutzer kann selbst Teil des Umgebungsmodells sein. Mehrere Benutzer können das selbe Umgebungsmodell verwenden.*

> **Anforderung 10: Mächtigkeit des Umgebungsmodells**
> *Das Umgebungsmodell soll die für die Anwendung benötigte Umgebungsinformation darstellen können.*

Dies ist eine grundlegende Anforderung, die sich auf die vom Umgebungsmodell modellierten Daten bezieht. Um diese Anforderung erfüllen zu können, muss zunächst der Anwendungsbereich untersucht werden.

> **Anforderung 11: Effiziente Verwaltung des Umgebungsmodells**
> *Das Umgebungsmodell soll in der Plattform effizient verwaltet werden können.*

Diese Anforderung folgt aus Anforderung 9 (Effizienz) und Anforderung 7 (Skalierbarkeit). Eine Darstellung, die von keiner möglichen Plattform effizient und skalierbar verwaltet werden kann, ist sicherlich nicht problemadäquat.

> **Anforderung 12: Erweiterbarkeit des Umgebungsmodells**
> *Das Umgebungsmodell soll um neue Umgebungsinformationen erweitert werden können.*

Dieser Fall tritt beispielsweise ein, wenn neue Anwendungen hinzukommen oder ein Datenanbieter neuartige Daten zur Verfügung stellen möchte (siehe auch Anforderung 8 (Offenheit)).

Anforderung 13: Einfachheit des Umgebungsmodells
Das Umgebungsmodell soll so einfach wie möglich gehalten sein.

Je einfacher das Umgebungsmodell ist, um so leichter kann es von Menschen verstanden werden. Dies ist wichtig, um Akzeptabilität zu ermöglichen: ein komplexes, schwer zu verstehendes Modell werden weder Datenanbieter noch Anwendungsentwickler gerne verwenden.

Everything has been said before, but since nobody listens we have to keep going back and beginning all over again. (A. Gide)

Kapitel 3: Verwandte Arbeiten

In diesem Kapitel wird der Stand der Wissenschaft, soweit für diese Promotion relevant, dargestellt. Ortsbezogene Anwendungen sind der Ausgangspunkt für diese Arbeit. Das Umgebungsmodell ist eine Art Ontologie, weswegen ontologiebasierte Ansätze kurz betrachtet werden. Aufgrund seiner räumlichen Struktur wird untersucht, ob sich bestehende Geodatenmodelle als Grundlage für das Umgebungsmodell eignen. Das World Wide Web stellt in seiner Offenheit und seiner globalen Skalierung ein Vorbild für die hier zu entwickelnde Lösung dar, die auch mit anderen Architekturen zur Informationsintegration verglichen wird. Schließlich werden existierende kommerzielle und prototypische Plattformen für ortsbezogene Anwendungen dargestellt und gezeigt, warum diese dem umfassenden Ansatz dieser Arbeit nicht gerecht werden.

3.1 Ortsbezogene Anwendungen

In der Forschung gibt es zahlreiche Arbeiten, die prototypisch orts- oder kontextbezogene Anwendungen entwickeln und untersuchen. Mit zunehmender Verbreitung von Mobilkommunikation gibt es auch erste kommerzielle Anwendungen, auf deren Stand in Kapitel 3.6 näher eingegangen wird.

Für die Entwicklung des Umgebungsmodells und der Plattform war es notwendig, eine Analyse und Klassifikation der Anwendungsdomäne ortsbezogener Anwendungen durchzuführen, deren Ergebnis in Kapitel 4 zu finden ist.

Einen Überblick über Eigenschaften verschiedener orts- und kontextbezogener Anwendungen geben [CK00] und [SAW94]. Angelehnt an die dortige Struktur werden die verschiedenen Arbeiten auch hier nach

Anwendungsfeldern aufgeführt (für eine genauere Abgrenzung der Gebiete siehe Kapitel 4.1.2): Ortsbezogene Informationssysteme, Navigation, Umgebungsannotation, instrumentierte Umgebungen und ortsbezogene Spiele, die häufig auch in den Bereich des sog. *Edutainment* fallen.

3.1.1 Ortsbezogene Informationssysteme

Ortsbezogene Informationssysteme liefern dem mobilen Benutzer Informationen über seine aktuelle Umgebung. Ein häufiger Anwendungsfall ist hier ein Tourist, der eine ihm unbekannte Stadt besucht, wie z.B. im *Cyberguide*-Projekt [LKA+96] oder im GUIDE-Projekt [CDM+00]. Dabei handelt es sich in der Regel um Multimedia-Informationen zu interessanten Plätzen, die je nach Ort des Benutzers angezeigt werden. Das *Deepmap*-Projekt [MZ00] verwendet hierzu ein detailliertes 3D-Stadtmodell der Stadt Heidelberg. Ein mobiles Informationssystem ist auch Teil des Projekts [DOM02], bei dem die Reiseplanung und die Kombination verschiedener Informationsquellen (Hotelreservierung, Navigation,...) im Vordergrund stehen. Neben touristischen Szenarien wurden auch andere Informationsräume verwendet, wie z.B. ein universitärer Campus ([KRC+03], [HFT+99]) oder Museen [RSZ+04].

Gemeinsam ist diesen Arbeiten, dass sie auf die Anwendung selbst und deren Benutzerinteraktion fokussieren. Die Umgebungsinformation, die sie benötigen, wird speziell für die Anwendung aufbereitet und in manchen Fällen bereits auf das Endgerät kopiert. Eine gemeinsame Nutzung von Umgebungsinformation durch verschiedene Anwendungen ist nicht vorgesehen. Da der Schwerpunkt in der vorliegenden Arbeit auf der Entwicklung einer Infrastruktur zur Verwaltung eines umfassenden Umgebungsmodells liegt, sind derartige Anwendungen jedoch als Anforderungslieferanten von großem Interesse.

3.1.2 Navigation

Das Berechnen einer kürzesten Wegstrecke von einem Start- zu einem Zielpunkt ist eine naheliegende und auch eine der nützlichsten ortsbezogenen Anwendungen. Die Voraussetzung hierfür ist ein digitales Verkehrsnetz, in dem Wegesuchalgorithmen verwendet werden können. Für den Bereich Fahrzeugnavigation auf Straßennetzen existieren schon seit einiger Zeit ausgereifte kommerzielle Lösungen. Sie basieren auf der Lokalisation durch GPS und verwenden meist ein festes Verkehrsnetzmo-

dell (z.B. auf CD). Teilweise werden auch Informationen dynamisch über Mobilfunknetze nachgeladen, um aktuelle Zustandsänderungen wie Staus oder Baustellen in die Verkehrswegeberechnung einzubeziehen.

Die elektronische Fahrplanauskunft des Öffentlichen Personennah- und Fernverkehrs ist eine Navigationsanwendung auf dem Bus- und Bahnnetz. Auch diese Anwendungen sind bereits weit verbreitet.

Im Bereich Fußgängernavigation gibt es bisher weitgehend Forschungsprojekte, die sich mit der Benutzerinteraktion und geeigneten Präsentationsformen beschäftigen, um z.B. schnell laufenden Fußgängern andere Informationen anzuzeigen als langsamen, wie im REAL/IRREAL-Projekt [BKW02]. Auch im Campus-Informationssystem des Mars-Projekts wurde eine Navigationskomponente für Innenräume entwickelt [HHT+01].

Für die vorliegende Arbeit können existierende Systeme als „bestehende Informationssysteme“ betrachtet werden, die aufgrund von Anforderung 4 integriert werden sollen. Darüber hinaus muss das Umgebungsmodell Netzwerkdaten der verschiedenen Verkehrsnetze enthalten. Besonders interessant ist dabei die Möglichkeit, auch multi-modale Navigation unterstützen zu können, also die Navigation über verschiedene Verkehrsnetze hinweg.

3.1.3 Umgebungsannotation

Nach Anforderung 2 (Ortsbasiertes Ablegen von Information) sollen Anwendungen ihre Umgebung auch annotieren können. Auch in diesem Bereich gibt es bereits Arbeiten, wie das *Stick-E-Notes*-Projekt [Pas97], das die *Post-It*-Metapher auf mobile Anwendungen überträgt, oder *comMotion* [MS00], mit dem mobile Benutzer *Todo*-Listen und Sprachnotizen an geographischen Orten zurücklassen können. Auch virtuelle Litfaßsäulen [LKR99], mit denen Webseiten mit Ortsbezug versehen werden können (eine frühe Arbeit im Nexus-Projekt), sehen die Möglichkeiten von Benutzerkommentaren vor.

3.1.4 Instrumentierte Umgebungen

Als instrumentierte Umgebung werden Räume, Gebäude oder allgemein Gebiete bezeichnet, die mit interaktiven Systemen, Sensoren und Aktoren ausgestattet sind und damit neuartige Anwendungen ermöglichen, bei

denen die Benutzerin direkt mit Alltagsgegenständen interagiert oder durch den sie umgebenden Raum in ihren Aufgaben unterstützt wird. Auch bei diesen Anwendungen spielt der Ort des Benutzers eine wichtige Rolle. Viele Forschungsprojekte in diesem Gebiet fokussieren auf die Benutzerinteraktion, wie z.B. i-land [SGH+99] und das *Aware Home* Projekt [KOA+99], oder auf die kontextbasierte Anpassung von Anwendungen (GAIA-Projekt, [RC00]). Ein zugrundeliegendes Kontextmodell oder eine globale, offene, skalierbare Infrastruktur ist in diesen Arbeiten nicht zu finden.

3.1.5 Spiele und „*Edutainment*“

Orthogonal zu den bisherigen Anwendungsbereichen ist der Bereich ortsbezogener (Lern-)Spiele. Da die Konzepte und Techniken mobiler, kontextbezogener Anwendungen noch neu sind, haben Anwendungen mit Spaßfaktor den Vorteil, dass Menschen eher bereit sind, sich darauf einzulassen. Damit können neue Anwendungskonzepte leichter erprobt werden, was dabei hilft, Konzepte und Methoden für „ernsthafte“ Anwendungen zu entwickeln. Zudem ist der Unterhaltungssektor ein nicht zu unterschätzender Wirtschaftsfaktor. Deswegen ist es durchaus sinnvoll, auch Spiele als Anwendungsgebiet in Betracht zu ziehen.

Lernspiele. Im GEIST-Projekt [HS04] wurde ein Prototyp für ein interaktives Lernspiel entwickelt, das es Benutzern erlaubt, die Geschichte Heidelbergs zur Zeit des 30jährigen Krieges hautnah zu erleben. Im Rahmen einer von Geistern erzählten Geschichte werden historische Fakten aber auch Legenden vermittelt. Auf diesem Wege werden Schüler und Erwachsene motiviert, sich mit Geschichte als Geschehenem auseinanderzusetzen.

In [BBD+03] wird ein mobiles Spiel vorgestellt, mit dem Venedig erkundet werden kann. Der Fokus liegt auf der Benutzer-Interaktion. Dieses Spiel benötigt keine Lokalisierung: Spielereignisse werden ausgelöst, wenn sich zwei Spieler begegnen.

Handel- und Kampfspiele. *Pirates*! [BFH+01] ist ein mobiles, ortsbezogenes Spiel, bei dem jeder Spieler als Kapitän sein imaginäres Piratenschiff durch eine virtuelle Inselgruppe steuert, in dem er sich bewegt. Dabei können z.B. Schätze gefunden werden, an Häfen Handel getrieben und mit anderen Schiffen gekämpft werden.

ARQuake! [PT02] ist eine mobile Umsetzung eines *Egoshooters*, bei dem der Spieler sich durch die Spielwelt bewegt und mit Hilfe einer Waffe versucht, Gegner zu treffen und Aufgaben zu lösen. Die Spielwelt wurde ersetzt durch die reale Welt, Monster und der derzeitige Waffen- und Lebensstatus der Spielfigur werden in eine halbdurchlässige Brille eingeblendet. Eine Aufgabe des Spiels ist es auch, Gegenstände aufzusammeln und versteckte Türen und Knöpfe zu entdecken.

Mobile Mixed Reality Games. In [FAB+03] werden Erfahrungen mit zwei ortsbasierten Spielen beschrieben, die als *mobile mixed reality games* bezeichnet werden, da Spielwirklichkeit und Realität miteinander verschmelzen. *Can you see me now?* fand zeitlich begrenzt an einem Wochenende im August 2001 statt. Hierbei gab es zwei Gruppen von Teilnehmern: die *Player* spielten von zu Hause aus und steuerten einen virtuellen Avatar durch die reale Stadt. *Runner* bewegten sich tatsächlich durch die Stadt. Die Position der *Runner* war für die *Player* zu sehen. Die Aufgabe der *Runner* war es, die Avatare der *Player* zu fangen, in dem sie sich auf ihre Position begaben. Im Spiel *Bystander* bewegt sich der Spieler selbst durch die Stadt und folgt den Spuren einer mysteriösen Person, von der er nur den Namen (*Uncle Roy*) und ein Bild kennt. Ein sog. *Online Performer* arbeitet mit ihm zusammen und führt ihn in seiner Suche, in dem er ihn zu verschiedenen Hinweisen auf *Uncle Roy* bringt, die z.B. in kurzen Video Filmen bestehen oder in einer Nachricht über einen öffentlichen Fernsprecher.

3.1.6 Abgrenzung

Wie gezeigt werden konnte, existieren bereits zahlreiche mobile, ortsbezogene Anwendungen, die zum Teil sehr vielversprechende und neuartige Konzepte umsetzen. Die meisten dieser Arbeiten fokussieren jedoch auf die Anwendungsentwicklung selbst, auf die Erprobung der Technologien oder auf die Benutzerinteraktion. Jede Anwendung wurde für sich selbst entwickelt, es gibt bisher kaum Systemunterstützung oder gemeinsame Verwendung von Umgebungsinformation. Es ist zu erwarten, dass bei steigender Anzahl von Anwendungsentwicklungen der Bedarf an einer Anwendungsplattform wächst. Hierfür sollen in dieser Arbeit die Grundlagen gelegt werden.

3.2 Ontologien

> *The principal advantage of a limited domain is ease of analysis, design, and implementation. Its weakness, however, is the difficulty of sharing and reusing data and programs in other applications. [...] To share knowledge with other applications, an ontology must be embedded within a more general framework. -- [Sow00], on building microworlds, page 53*

Der Begriff der Ontologie stammt aus dem Griechischen und steht für die „Lehre vom Sein“ (onto-logos). Er entstammt der philosophischen Tradition: schon früh haben die Menschen daran geforscht, wie das intuitive gemeinsame Verständnis dessen, was in der Welt ist, festgehalten und ausgetauscht werden kann. Eine Ontologie beschreibt Entitäten, deren Eigenschaften, Beziehungen zu einander und Regeln, aus denen sich weiteres Wissen ableiten lässt.

Auch in der Informatik gibt es zahlreiche Ontologien: z.B. die Wissensbasis eines Expertensystems, das Schema einer Datenbank, das Begriffslexikon einer Spezifikation oder ein Datenaustauschstandard für Geschäftsdaten, um nur ein paar zu nennen. Sie werden nicht immer als Ontologie bezeichnet, doch im Grunde sind sie genau das: ein gemeinsames Verständnis über Begriffe und Bedeutungen.

Diese Ontologien unterscheiden sich: nach dem Grad ihrer Formalisierung, der Mächtigkeit ihrer Beschreibungssprache, dem beschriebenen Weltausschnitt und vor allem: ihrem Zweck, der die anderen Faktoren bestimmt.

Im Folgenden werden deswegen zunächst verschiedene Verwendungsmöglichkeiten von Ontologien in der Informatik aufgezeigt. Danach folgt ein kurzer Exkurs über den Entwurf von Ontologien nach [UG96], da auch die Entwicklung des Umgebungsmodell in dieser Arbeit im wesentlichen diesem Vorgehen folgt. Es wird ein Überblick über existierende Ontologien und Beschreibungssprachen gegeben. Das Kapitel schließt mit der Abgrenzung der vorliegenden Arbeit zu diesem Gebiet.

3.2.1 Verwendung von Ontologien

[BCM+03] identifiziert in seinem Rahmenwerk für die Klassifikation von Ontologie-Anwendungen drei Hauptbereiche, die auch schon in [UG96] aufgeführt werden: Kommunikation (zwischen Menschen), Interoperabilität (zwischen Computersystemen) und System-Entwurf.

Kommunikation. Ohne implizite Ontologien wäre Kommunikation zwischen Menschen überhaupt nicht möglich. In einigen Fällen ist es jedoch vorteilhaft, wenn sie explizit aufgeschrieben werden. So normiert z.B. das Begriffslexikon einer Spezifikation das Weltverständnis zwischen Kunden und Entwicklern, denn explizite Definitionen verhindern Inkonsistenzen und Missverständnisse und sorgen schon bei ihrer Entwicklung für ein klareres Verständnis der Dinge. Für solche Ontologien reicht meist eine eindeutige, aber informelle Beschreibungssprache (z.B. formalisierte natürliche Sprache) aus.

Interoperabilität. Interoperabilität ist die Zusammenarbeit zwischen unabhängigen Computersystemen. Ein System nutzt Dienste des anderen, die Systeme tauschen Daten aus oder nutzen die selben Ressourcen. Auch hierfür ist ein gemeinsames Verständnis der Daten, Dienste oder Ressourcen notwendig. Da die Systeme meist eine unterschiedliche Interpretation und Darstellung verwenden, dient eine Ontologie hier als Interlingua: die Systeme übersetzen ihre Darstellung in die gemeinsame Ontologie. Diese Übersetzung kann auch von externen Systemen, sog. Wrappern, vorgenommen werden, so dass das eigentliche System unangetastet bleibt.

Ein Beispiel für diese Verwendung von Ontologien sind Datenaustauschstandards für bestimmte Anwendungsbereiche, wie z.B. Geodatenmodelle, auf die im Kapitel 3.3 näher eingegangen wird.

System-Entwurf. Der Einsatz einer Ontologie bietet einige Vorteile für den System-Entwurf, wie z.B.:

- Wiederverwendbarkeit: Die Ontologie kodiert das zentrale Weltwissen eines Systems, das dann auch von anderen Systemen im selben Anwendungsbereich verwendet oder mitverwendet werden kann.
- Suche: Die Ontologie kann als Wissenbasis oder als Suchindex für Information verwendet werden. Das *Semantic Web* ist hierfür ein Beispiel (siehe Kapitel 3.5.4).
- Verlässlichkeit: Eine formale Ontologie kann automatisch auf Konsistenz geprüft werden.
- Spezifikation: Die Ontologie kann den Prozess der Anforderungsanalyse und Spezifikation eines Systems unterstützen.

3.2.2 Entwicklung einer Ontologie

Im Folgenden wird die Entwicklung einer Ontologie beschrieben. In diesem Bereich gibt es keine allgemeingültige Methode, sondern verschiedene Gruppierungen haben ihre eigenen Verfahren entwickelt [PB99]. Hier wird nun die Verfahrensweise von [UG96] vorgestellt, die aus den Erfahrungen des Entwicklungsprozesses der *Enterprise Ontology* entstand, da sie sehr gut allgemeine Arbeitsphasen und Problemstellungen identifiziert. Sie wird hier zur Entwicklung dieser Arbeit in Beziehung gesetzt. Die Methode enthält mehrere Phasen, die nacheinander, teilweise parallel oder iterierend ablaufen können.

Zweck und Umfang definieren. Zu Beginn sollte geklärt werden, zu welchem Zweck die Ontologie entwickelt wird. Denn davon hängt in den weiteren Schritten ab, welcher Grad des Formalismus in der Beschreibungssprache notwendig ist, und welchen Umfang die Ontologie haben sollte. In dieser Arbeit wird der Zweck in Kapitel 2 beschrieben: eine Plattform für kontextbezogene Anwendungen soll durch ein Umgebungsmodell (die Ontologie) unterstützt werden. Der Umfang wird durch die Anwendungsdomäne abgesteckt (Kapitel 4.1).

Ontologie erfassen. In dieser Phase werden die wichtigsten Konzepte und Entitäten der Ontologie und deren Beziehungen identifiziert, in einer eindeutigen Textdefinition aufgeschrieben und darüber eine Einigung erzielt. Dabei wird ein *middle-out*-Vorgehen vorgeschlagen: es wird weder von den allgemeinsten Begriffen ausgegangen (*top-down)* – die vielleicht noch gar nicht gefunden wurden – noch von einer unzusammenhängende Sammlung der speziellsten Begriffe (*bottom-up*) – die zu Beginn die Entwickler mit zu vielen Details erschlagen würden, sondern von den zentralen Begriffen, die dann nach und nach verallgemeinert und spezialisiert werden können.

Eine zentrale Rolle hierbei können die sog. *motivating scenarios* spielen. Dies sind Anwendungsfälle für die Ontologie, die als Prüfstein für die Vollständigkeit dienen können: es können informelle (An-)Fragen abgeleitet werden, die aus der Ontologie heraus beantwortet werden können sollen.

In dieser Arbeit finden sich Szenarien verschiedener ortsbezogener Anwendungen in Kapitel 4.2.

Ontologie kodieren. Für manche Verwendungsarten (z.B. Begriffslexikon) von Ontologien ist eine Textdefinition ausreichend. Soll die Ontologie jedoch von Maschinen verarbeitet werden, so muss sie kodiert werden, sprich, in einer strikten und formalisierten Darstellung erfasst werden.

Hierzu muss zunächst eine formale Beschreibungssprache ausgewählt werden, um die Ontologie selbst zu beschreiben. Daraus leitet sich häufig die Sprache ab, in der dann die Instanzen der Ontologie, die eigentlichen Daten, beschrieben werden. In Kapitel 3.2.3 wird auf verschiedene Beschreibungssprachen für Ontologien eingegangen.

Ausgehend von den Ergebnissen der vorherigen Phase werden nun die Konzepte und Entitäten formal spezifiziert und dabei auf Vollständigkeit und Widerspruchsfreiheit geprüft.

Nach [UG96] können zu einer solchen Ontologie-Spezifikation nun formale Axiome und ein Vollständigkeitstheorem hinzugefügt werden, um die gewünschten Eigenschaften der Ontologie formal zu überprüfen. Aus den informellen Anfragen aus der vorherigen Phase werden nun sog. *competency questions* abgeleitet, anhand derer zusätzlich überprüft werden kann, ob die Ontologie ihren Zweck erfüllt und die erwünschten Aussagen treffen kann.

In dieser Arbeit wurde auf den letzten Schritt verzichtet. Das Umgebungsmodell benötigt für die Erfüllung seiner Anforderungen keinen Formalisierungsgrad, der solche Überprüfungen ermöglichen würde.

Bestehende Ontologien integrieren. Während einer oder beider der obigen Phasen stellt sich die Frage, wie und ob bestehende Ontologien integriert werden können und sollen. Nach [UG96] ist dies im Allgemeinen ein schwieriger Prozess, zu dem noch wenig Methoden und Werkzeuge existieren.

In dieser Arbeit wurden bestehende Geodatenstandards daraufhin untersucht, wie sie im Rahmen des Umgebungsmodells eingebunden oder direkt unterstützt werden können. Die Untersuchung findet sich in Kapitel 3.3, die tatsächliche Einbindung in das Umgebungsmodell wird in Kapitel 6 beschrieben. Zudem wurden Ontologien bzw. Datenmodelle, die in anderen Systemen entstanden, bei der Integration in die Nexus-Systemplattform auf das Umgebungsmodell abgebildet (Kapitel 7.2).

Evaluieren. [UG96] schlägt vor, für die Evaluation von Ontologien Techniken aus dem Bereich des *knowledge sharing* zu verwenden und anzupassen. Eine gute Definition davon gibt z.B. [GJP95]:

> „to make a technical judgement of the ontologies, their associated software environment, and documentation with respect to a frame of reference. [...] The frame of reference may be requirements specifications, competency questions, and/or the real world."

Das Bezugssystem des Umgebungsmodells und der dazugehörigen Plattform in dieser Arbeit besteht demnach zum einen aus den Anforderungen aus Kapitel 3, den Anwendungsfällen aus Kapitel 4.2 und natürlich der realen Welt. Letztendlich könnten nur Endbenutzer-Studien die Übereinstimmung des Umgebungsmodells mit der realen Welt überprüfen, die jedoch den Rahmen dieser Arbeit sprengen würden. Stattdessen wird es anhand von Szenarien der Anwendungsentwicklung evaluiert (Kapitel 8).

Dokumentieren. Eine Dokumentation der Konzepte und Annahmen einer Ontologie ist wichtig, damit verschiedene Leute sie verwenden können. [UG96] empfiehlt dafür Ontolingua mit der Unterstützung des KSL Ontology Editors.

Das Umgebungsmodell ist zum einen über Annotationen seines Klassenschemas dokumentiert (mit Unterstützung von XML-Editoren), zum anderen durch die vorliegende Arbeit.

3.2.3 Ontologiebeschreibungssprachen

Grundsätzlich existieren zwei verschiedene Ansätze, um Ontologien zu formalisieren: frame-basierte und logik-basierte (auch unter dem Begriff *description logic* bekannt) [PB99]. Logik-basierte Ansätze wollen durch die strikte Modellierung von Axiomen und Regeln aus bekanntem Wissen neues Wissen ableiten (Inferenz). Der Fokus liegt auf einer expliziten Semantik und auf Reasoning, die Entscheidbarkeit des Logik-Systems ist eine wichtige Eigenschaft. Bei den frame-basierten Ansätzen dagegen geht es darum, alle relevante Information über einen Kontext in einem Objekt zu bündeln, anstatt sie über mehrere Axiome zu verteilen. Ein Objekt (oder *Frame*) besitzt dabei mehrere *Slots*, um seine Eigenschaften

zu beschreiben [BCM+03]. Das Ziel dieses Ansatzes ist es demnach, möglichst viel Wissen aus der Welt zu modellieren, auf Kosten der Entscheidbarkeit.

Es gibt zahlreiche Beschreibungssprachen für Ontologien, und es würde den Rahmen dieser Arbeit sprengen, sie alle zu betrachten. Im Folgenden wird deswegen nur ein kurzer Überblick gegeben. 1999 waren nach [PB99] die repräsentativsten Vertreter von Ontologie-Beschreibungssprachen Ontolingua [Gru93] und F-Logic [KLW95] auf der Seite der framebasierten Systeme, Loom [Mac91] und CycL [LG90] bei den logik-basierten Ansätzen. Seitdem hat in diesem Forschungsbereich jedoch eine Konsolidierung stattgefunden. Mit der Vision des *Semantic Web* (siehe Kapitel 3.5.4) entstanden zunächst weitere XML-basierte Sprachen wie RDF, DAML und OIL, bis diese in einen von vielen Gruppierungen getragenen W3C Standard übergingen, die OWL Web Ontology Language [OWL].

OWL besteht aus drei Teilsprachen, um den Anforderungen der beiden Ansätze zu entsprechen: OWL DL, OWL Lite und OWL Full [HPH03]:

OWL DL. OWL DL (*description logic*) besitzt eine einfache Syntax und ermöglicht entscheidbare Inferenz. Diese Version von OWL kann wahlweise in frame- oder logikbasierter Art geschrieben werden.

OWL Lite. OWL Lite, eine syntaktische Teilmenge von OWL DL, besitzt eine noch einfachere Syntax und eine lenkbarere Inferenz; dafür schränkt es die Modellierungsmöglichkeiten und Inferenzmöglichkeiten ein.

OWL Full. OWL Full ermöglicht Kompatibilität zu seinem Vorgänger RDF [RDF] und bietet eine semantische und syntaktische Erweiterung dazu. Es enthält auch OWL DL, geht aber über den üblichen Rahmen der *description logic* hinaus. OWL Full ist unentscheidbar, und die abstrakte Syntax von OWL DL ist unpassend. Die offizielle OWL Austauschsprache ist RDF/XML.

3.2.4 Ontologien für kontextbezogene Systeme

Es gibt bereits ontologiebasierte Ansätze für kontextbezogene Anwendungen, von denen zwei hier vorgestellt werden sollen.

Die Context Broker Architektur (CoBrA) [CFJ04] verwendet die Ontologie COBRA-ONT. Es enthält Klassen über physische Orte, Agenten (sowohl menschliche als auch Software), den Ortskontext von Agenten und ihre Aktivität. CoBrA zielt auf Szenarien, in denen Menschen auf einem universitären Campus in einem Meeting zusammenkommen. Für dieses Szenario werden 41 OWL Klassen und 36 *properties* benötigt, um die Wissensbasis zu modellieren.

CONON [WZG+04], die „CONtext ONtologie“ besteht aus einer oberen Kontextontologie, die allgemeine Konzepte über Kontext modelliert, und bietet Erweiterungsmöglichkeiten für anwendungsspezifische Ontologien über eine Hierarchie. In einem Prototyp wurde eine spezifische Ontologie für eine intelligente Heimumgebung implementiert, die 197 OWL Klassen benötigte.

Weitere Ontologie-Ansätze wie z.B. [CYC] enthalten ebenfalls Anteile von Kontextinformation, sind jedoch nicht darauf fokussiert und damit häufig zu allgemein für das Problem. Eine zu komplexe Ontologie, die noch zahlreiche weitere Aspekte modelliert, widerspricht jedoch der Anforderung 13 (Einfachheit des Umgebungsmodells).

3.2.5 Die Ontologie-Ebenen nach Andrew U. Frank

[Fra03] unterteilt räumliche Ontologien in fünf Ebenen, die für die Einordnung dieser Arbeit sehr nützlich sind (siehe Tabelle 1).

Die Ebene 0 ist die Ontologie der physischen Realität. Sie basiert auf der Annahme, dass es genau eine reale Welt gibt. Also kann für jede Eigenschaft in der Welt zu einem gegebenen Raum-Zeit-Punkt ein einzelner Wert bestimmt werden.

Die Ebene 1 enthält Beobachtungen der Realität. Dies ist die erste Ebene, die von Rechnersystemen und auch Menschen erfasst werden kann. Ein Wert wird hier nun mit einem bestimmten Beobachtungstyp (*observationType*) festgelegt. Dieser Typ bestimmt die Messwertskala (z.B. nominal, rational,...) und die Einheit des Messwerts (z.B. Meter oder Sekunden). Für räumliche Werte muss auch ein Referenzsystem gegeben werden. Beobachtungen der Realität sind immer ungenau – eine theoretische Grenze ist die Heisenbergsche Unschärferelation, aber in der Regel haben die Ungenauigkeiten ihren Grund in der verwendeten Messtechnik oder Fehlern.

Tabelle 1: Die fünf Ebenen räumlicher Ontologien nach [Fra03]

Ontologie Ebene 0: Physische Realität *reality :: world –> property –> spacePoint –> timePoint –> value*
Ontologie Ebene 1: Beobachtungen der Realität *observation :: world –> observationType –> location –> value*
Ontologie Ebene 2: Welt der Objekte *observation :: id –> time –> observationType –> value*
Ontologie Ebene 3: Soziale Realität *getname :: object –> name* *findObject :: name –> environment –> object*
Ontologie Ebene 4: Kognitive Agenten *rules used or deduction*

In der Ebene 2 werden einzelne Beobachtungen zu Objekten gruppiert. Objekte sind dadurch definiert, dass sie gemeinsame Eigenschaften haben. Die Ebene enthält nur physische Objekte, also Dinge, die in der physischen Welt existieren und beobachtet werden können. Physische Objekte haben eine geometrische Grenze in der Welt, die sich jedoch über die Zeit hinweg ändern kann (z.B. bei Sanddünen oder Flüssigkeiten).

Bisher enthielten die Ontologie-Ebenen Daten, die man als objektiv bezeichnen kann. Eine Gruppe von Menschen, die mit Messgeräten eine bestimmte Umgebung erfassen soll, kann sich über die geometrischen Grenzen und physischen Eigenschaften von Objekten einig werden. Anders ist es in Ebene 3, der soziale Realität. Nun kommen Interpretationen und Benennungen der Gesellschaft hinzu. Die Eigenschaften dieser Ebene entstammen meist einem administrativen, rechtlichen oder institutionellen Kontext. Auch die Namen von Objekten gehören zu dieser Ebene, da sie kulturell bestimmt sind. Für viele wichtige Objekte gibt es Namen, die jedoch aufgrund kultureller Disposition und Sprachgebrauch unterschiedlich sind.

Tabelle 2: Einordnung in die fünf Ebenen

	Ontologie Ebene 0: Physische Realität	
ontologiebasierte Ansätze	Ontologie Ebene 1: Beobachtungen der Realität	Umgebungsmodell
	Ontologie Ebene 2: Welt der Objekte	
	Ontologie Ebene 3: Soziale Realität	
	Ontologie Ebene 4: Kognitive Agenten	

Schließlich bildet die Ebene 4 die Regeln ab, die von kognitiven Agenten (Softwaresystemen oder Menschen) benutzt werden, um zu Entscheidungen über die bisher modellierte Welt zu gelangen. Diese Regeln sind meist in Anfragesprachen, Anwendungen oder auch Inferenzmaschinen kodiert.

3.2.6 Abgrenzung

Die fünf Ebenen können genutzt werden, um die vorliegende Arbeit gegenüber ontologiebasierten Ansätzen abzugrenzen (siehe Tabelle 2). Bestehende ontologiebasierte Ansätze decken alle vier Ebenen ab: sie zielen darauf ab, aus vorhandenem Wissen weiteres Wissen abzuleiten und benötigen dafür eine detaillierte Darstellung der unteren Ebenen. Da die Interpretation der Regeln jedoch eine rechenintensive Operation mit hoher Komplexität ist, sind die Szenarien solcher Ansätze häufig eher klein (siehe Kapitel 3.2.4). Dies widerspricht jedoch der Anforderung 7 (Skalierbarkeit), da auch Szenarien mit sehr vielen Objekten unterstützt werden sollen.

Das Umgebungsmodell überlässt deswegen die Interpretation der Modelldaten der Anwendung und modelliert hauptsächlich Ebene 1 bis Ebene 3. Damit ist es möglich, innerhalb einer sozialen Wirklichkeit zu einer Einigung zu kommen und eine konsistente Weltsicht zu bilden. Die gemeinsame soziale Wirklichkeit kann dadurch gewährleistet werden, dass nur eine bestimmte Anwendungsklasse, nämlich ortsbasierte

Anwendungen, unterstützt werden sollen. Dies bedeutet, dass bestimmte Sachverhalte für die Anwendungsklasse fest definiert werden können, die in anderen Anwendungsklassen eine andere Bedeutung hätten.

Ebene 4 wird in der vorliegenden Arbeit durch die Anwendungen selbst abgedeckt. Ein erster Ansatz, wie Inferenzen und Regeln trotzdem wiederverwendbar für mehrere Anwendungen durch die Plattform unterstützt werden können, findet sich im Ausblick in Kapitel 9.2.

Zusammenfassend kann man sagen, dass es sich bei dem Umgebungsmodell durchaus um eine Ontologie handelt, die jedoch weniger interpretative Elemente enthält als ontologiebasierte Ansätze; die Interpretation und die Ableitung von Situationen wird der Anwendung überlassen.

3.3 Geodatenmodelle

Wie wir in Kapitel 4.3 sehen werden, benötigen viele ortsbezogene Anwendungen Informationen aus (digitalen) Karten, im Folgenden als Geodaten bezeichnet. Solche Daten werden schon lange in Geoinformationssystemen (GIS) erhoben, verwaltet und analysiert und sind für viele Anwendungen essentiell (z.B. Stadt- und Regionalplanung, Umwelt- und Naturschutz, demographische Analysen, etc).

Bisher wurden in den verschiedenen GIS-Bereichen jeweils eigenständige Modelle erhoben, die sich in vielen Aspekten mitunter stark unterscheiden: z.B. geographische Ausdehnung, Detailgrad, Genauigkeit, Datenschema und Semantik, unterstützende Systeminfrastruktur, Präsentation, Codierung oder schlicht Inhalt. Allerdings wurde gerade im Zeitalter der Vernetzung zwischen Computersystemen erkannt, dass es wünschenswert ist, Brücken zwischen diesen zahlreichen Insellösungen zu bauen, da die Erhebung und Fortführung von Geodatenmodellen sehr teuer ist und die gemeinsame Verwendung die Effizienz steigern kann.

Aus diesem Bestreben heraus werden verschiedene Initiativen zur Standardisierung voran getrieben:

- Auf internationaler Ebene ist das zentrale Gremium das OpenGIS Konsortium (OGC).
- National standardisiert die Arbeitsgemeinschaft der Vermessungsverwaltungen der BRD (AdV) die amtlichen Daten.
- Daneben gibt es Industriestandards, die sich aus der Marktposition der entwickelnden Firmen ergeben.

Für diese Arbeit ist es von Interesse, die verschiedenen Ergebnisse dieser Bestrebungen zu untersuchen. Dabei soll aufgezeigt werden, welche Standards ganz oder teilweise für die Spezifikation des Umgebungsmodells verwendet werden können und welche Grenzen die jeweilige Modellierung aufweist. Im Folgenden werden nun die wesentlichen Geodatenstandards aus diesen Initiativen kurz vorgestellt: die *Geographic Markup Language (GML)* des OGC, welche u.a. die Grundlage für den Standard *OpenLS* legt, die nationalen Standards *ATKIS, ALK* und das *AAA* Referenzmodell und den Industriestandard *Geographic Data Files (GDF).* Zum Abschluss des Kapitels wird noch kurz auf den Industriestandard *Physical Markup Language* eingegangen, der eigentlich kein Geodatenstandard ist, sondern die Beschreibung physischer Objekte (und deren Einbettung in logistische und Produktionsprozesse) ermöglicht. Da diese Objekte jedoch auch einen Ortsbezug haben können, muss untersucht werden, ob sich PML zur Darstellung des Umgebungsmodells eignet.

3.3.1 Geographic Markup Language (GML)

Die *Geographic Markup Language (GML)* [GML] ist eine Implementierungsspezifikation des OpenGIS Konsortiums. Sie folgt der abstrakten Spezifikation des *OpenGIS Features* (Fachobjekt), das grundlegende Eigenschaften der Repräsentation eines Realweltobjekts festlegt.

Der Zustand eines Fachobjekts wird über Tripel aus {Name, Typ, Wert} dargestellt. Welche Zustände ein Fachobjekt haben kann, legt der *FeatureType* fest. Fachobjekte können zu *FeatureCollections* gruppiert werden und verschiedenste Granularitäten aufweisen, von einem einzelnen Bildpunkt einer Satellitenaufnahme bis zu einer ganzen Stadt. Die Semantik der Fachobjekte wird dabei weder von der abstrakten Spezifikation noch von der Implementierungsspezifikation GML festgelegt: hierfür muss ein Applikationsschema definiert werden, das innerhalb einer Informationsgemeinschaft festlegt, welche Fachobjekte existieren, wie diese heißen und welche Zustände sie annehmen können. *OpenLS* (s.u.) ist ein solches Applikationsschema für ortsbasierte, mobile Anwendungen, das ebenfalls im Rahmen des OGC spezifiziert wurde.

Das Ziel von GML ist die Modellierung, der Austausch und die Speicherung raumbezogener Daten, basierend auf XML [XML]. Die Spezifikation besteht aus unterschiedlichen XML Schemata [XML Schema], die spezi-

elle Datentypen für die Spezifikation von Fachobjekten liefern, z.B. für Geometrie, Topologie, temporale Aspekte, etc. Diese Schemata können gemeinsam oder auch einzeln in Applikationsschemata verwendet werden.

An dieser Stelle stellt sich die Frage, ob das Schema des Umgebungsmodells nicht als GML Applikationsschema spezifiziert werden kann. Hierzu wird zunächst ein Blick auf ein existierendes Applikationsschema für ortsbezogene Anwendungen geworfen, um dann die Gemeinsamkeiten und Unterschiede zum Ansatz dieser Arbeit aufzuzeigen.

3.3.2 *OpenLS*

OpenLS ist eine Initiative innerhalb des OGC, die von mehreren Herstellern und Anbietern ortsbezogener Dienste (*location-based services*, kurz LBS) vorangetrieben wird. Das Ziel der Initiative ist es, eine Austauschsprache für ortsbezogene Daten und Dienste zu standardisieren sowie eine Referenzarchitektur für den Datenaustausch festzulegen (die sich an bestehenden Plattformen für ortsbezogene Dienste orientiert, siehe Kapitel 3.6.2). Das Datenmodell von *OpenLS* ist sehr einfach und besteht aus sogenannten Abstrakten Datentypen (ADT). Interessant ist die Tatsa-

Tabelle 3: Abstrakte Datentypen im *OpenLS* Standard

Datentyp	wird verwendet für
Position ADT Point	Koordinate in einem bekannten Koordinatensystem
Address ADT	Straßenadresse oder Kreuzung
Point of Interest (POI) ADT	wichtiger Ort (an dem z.B. Produkte oder Dienste zu finden sind)
Area of Interest (AOI) ADT	Polygon, Rechteck oder Kreis, der als Suchmuster verwendet wird
Location ADT	**Position, Address** oder **Point of Interest**
Map ADT	Darstellung von Karten (mit Routen oder *POIs*)
Route Summary ADT	Beschreibung einer Route
Route Geometry ADT	Geometrie einer Route
Route Maneuvers ADT	allgemeine Navigationsanweisungen für eine Route
Route Directions ADT	detaillierte Navigationsanweisungen für eine Route

che, dass *OpenLS* den GML-Standard nur dazu verwendet, die räumlichen Geometrien der Objekte zu beschreiben. Die *ATDs* in *OpenLS* sind keine Fachobjekte im Sinne von GML. Tabelle 3 gibt eine Übersicht über die unterstützten Datentypen. Diese sind sehr allgemein gehalten (z. B. stellt „Point of Interest“eine Sehenswürdigkeit oder anderweitig interessanten Punkt dar, ohne anzugeben, worum es sich handelt) und decken dabei nur die für den speziellen Anwendungsfall heute gebräuchlicher mobiler, ortsbasierter Anwendungen notwendigen Informationen ab (so fehlen mobile Objekte oder technische Objekte der Infrastruktur).

Der Ansatz dieser Arbeit geht jedoch sehr viel weiter: wie in den Kapiteln 4 und 6 zu sehen sein wird, soll im Umgebungsmodell ein weitaus detaillierteres Modell des realen und digitalen Kontextes einer Anwendung repräsentiert werden, um auch zukünftige Anwendungen und weitergehende Funktionen der Systemplattform zu unterstützen. Maßgeblich ist dabei unter anderem das Ziel, verschiedene lokale Modelle zu einem globalen Modell zu föderieren, was Konzepte zur Darstellung mehrfach repräsentierter Objekte erfordert.

Es sollte jedoch durch einen einfachen Transformationsschritt möglich sein, Daten im *OpenLS*-Standard in das Umgebungsmodell einzubringen.

3.3.3 Abgrenzung: Das Umgebungsmodell als GML Applikationsschema?

Auf den ersten Blick könnte das Umgebungsmodell als GML Applikationsschema modelliert werden: die für die Anwendungsdomäne notwendigen Informationen würden als Fachobjekte spezialisiert, der Ortsbezug könnte über die entsprechenden Schemata von GML beschrieben werden und die Verwendung eines internationalen Standards müsste Vorteile bei der Verfügbarkeit von Werkzeugen und Softwarekomponenten haben, die Daten dieses Formats verarbeiten können.

Das *OpenGIS* Referenzmodell fordert einige Eigenschaften für Fachobjekte. Wenn man diese mit gewünschten Eigenschaften von Objekten des Umgebungsmodells vergleicht, so stellt man fast, dass viele dieser Eigenschaften auch dort gewollt sind. So haben Fachobjekte z.B. eine globale, eindeutige, persistente ID (entspricht Anforderung 3 (ID-basierter Zugriff auf Information)) oder können durch Text beschrieben werden, wobei die Beschreibung nicht eineindeutig ist (entspricht dem Namen eines Objekts). Allerdings werden auch Eigenschaften gefordert, die auf die

Objekte des Umgebungsmodells nicht zutreffen: zu einem Realweltobjekt soll innerhalb einer Informationsgemeinschaft nur ein Fachobjekt existieren. Die Anforderung 4 (Integration bestehender Informationsdienste) führt jedoch unter anderem dazu, dass verschiedene Anbieter von Umgebungsmodelldaten verschiedene Modellierungen vornehmen. Wenn nun das umfassende Umgebungsmodell als Informationsgemeinschaft aufgefasst werden soll, so muss es flexiblere Modellierungskonzepte für die Darstellung mehrfach repräsentierter Objekte anbieten, wie z.B. Mehrfachtypen. Diese sind in GML-Applikationsschemata nicht vorgesehen: der Typ eines Fachobjekts wird über den Tag des umschließenden XML-Elements festgelegt und ist damit nicht mehrfach spezifizierbar. Auch gibt es im Umgebungsmodell keine komplexen Fachobjekte, die aus mehreren einfachen Fachobjekten bestehen: aufgrund der Anforderung 13 (Einfachheit des Umgebungsmodells) wurde darauf verzichtet, Objekte zu schachteln. Es hat sich herausgestellt, dass der räumliche Bezug für räumliche Enthaltensseinsbeziehungen (z.B. Räume in einem Gebäude) ausreicht, und dass andere Beziehungen auch über Relationen abgebildet werden können.

Es ist zu vermuten, dass aus ähnlichen Gründen (Einfachheit, keine vollständige Abbildbarkeit) auch der *OpenLS* Standard keine „echten“ GML-Fachobjekte spezifiziert.

Für die Darstellung des Ortsbezugs wurde im Umgebungsmodell daher ein ganz ähnlicher Ansatz wie beim *OpenLS* Standard gewählt: die Objekte werden mit XML Schema-Technologie beschrieben und für den Ortsbezug werden die entsprechenden GML-Schemata importiert. Zur Repräsentation von Mehrfachvererbung und Mehrfachtypen (die in XML Schema nicht vorgesehen ist) musste jedoch ein erweiterter Ansatz verwendet werden, der in Kapitel 6.8 näher erläutert wird.

Die Hoffnung, bei der Verwendung von GML auf bestehende Werkzeuge zurückgreifen zu können, erfüllte sich nicht: zu dem Zeitpunkt, als wesentliche Teile der Systemplattform zur Verwaltung des Umgebungsmodells entwickelt wurden, unterstützten weder räumliche Datenbanken noch geeignete Softwarebibliotheken den GML Standard. Der SQL99 Standard implementiert zwar die *Simple Feature Specification [SFS]*, die abstrakte Spezifikation von Geometrien, die auch GML zugrunde liegt, allerdings in einer anderen Repräsentation (dem *Well Known Text* Format). Somit wurde für das Umgebungsmodell ein dualer Ansatz gewählt:

Geometrien können sowohl in WKT als auch in GML repräsentiert werden, und es wurde eine Bibliothek entwickelt, die beide Formate unterstützt und ineinander umwandeln kann [SHG+04].

Zusammenfassend ist zu sagen, dass das Umgebungsmodell zwar einige Eigenschaften eines GML Applikationsschemas aufweist und mit Abstrichen vermutlich auch als solches repräsentiert werden könnte. Für diese Arbeit wurde jedoch ein eigener, verwandter Ansatz gewählt, auch um den besonderen Eigenschaften, die das Umgebungsmodell aufgrund seiner Zielsetzung aufweist, Rechnung zu tragen.

3.3.4 ATKIS, ALK und das AAA Referenzmodell

Das Amtliche Topographisch-Kartographische Informationssystem (ATKIS) ist ein von der Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV) ins Leben gerufenes Basisinformationssystem für topographische Geodaten, wie z.B. Vegetation, Siedlungen, Verkehr, etc. Es zielt auf keine bestimmte Anwendungsdomäne, sondern soll für verschiedene Anwendungen die notwendigen Geodaten bereitstellen. Das Datenmodell in ATKIS ist hierarchisch aufgebaut, unterstützt jedoch keine Vererbung.

Das Automatisierte Liegenschaftskarte (ALK) dient der Darstellung von kartographischen Objekten (Gebäuden, Straßen,...). Die ALK ist in verschiedenen thematischen Ebenen organisiert, die übereinander gelegt werden können. Die kartographischen Objekte können mit Fachdateien verknüpft werden, die teilweise auch nicht öffentlich zugänglich sind (z.B. Eigentumsverhältnisse). Die Semantik der Daten in ALK ist dabei nicht allgemein festgelegt: verschiedene Bundesländer definieren jeweils unterschiedliche Bezeichner und Bedeutungen, einen bundeseinheitlichen Standard gibt es nicht.

Zusammen mit dem neu entstandenen Automatisierten Festpunkt System (AFIS) werden diese nationalen Standards derzeit in das gemeinsame AAA Referenzmodell überführt. Damit soll als Austauschsprache für Geometrien in Zukunft GML verwendet werden, damit die Daten zumindest teilweise OGC-konform werden. Dieser Prozess ist jedoch noch nicht abgeschlossen.

Keiner der hier vorgestellten Standards eignet sich als Modellierungsgrundlage für das Umgebungsmodell. ATKIS und ALK sind Textformate, die nicht auf XML basieren. Es ist nicht vorgesehen, sie zu erweitern, so dass ein Konflikt mit der Anforderung 12 (Erweiterbarkeit des Umgebungsmodells) besteht. Und sobald weitere, nicht vorhergesehene Anwendungen hinzukommen, kommt es zu einem Konflikt mit der Anforderung 6 (Erweiterbarkeit bezüglich neuer Anwendungen). Auch die Modellierungskonzepte selbst sind nicht flexibel genug, um den sonstigen Anforderungen an das Umgebungsmodell gerecht zu werden (ATKIS unterstützt z.B. keine Vererbung).

Daten in den oben genannten Formaten bilden jedoch eine wichtige Grundlage für lokale Umgebungsmodelle. Sie lassen sich in die Darstellung des Umgebungsmodells transformieren, was z.B. am Beispiel von Straßendaten exemplarisch gezeigt werden konnte [VGH+02].

3.3.5 Geographic Data Files (GDF)

Das *Geographic Data Files* Format (GDF) ist ein Standard zur Realisierung von Fahrzeugnavigationsanwendungen. Er wurde ursprünglich von der Industrie entwickelt, hat aber inzwischen den Status eines ISO-Standards erlangt.

GDF definiert einfache und komplexe *Features*, die aus einfachen aufgebaut werden. Die Eigenschaften von *Features* werden durch Attribute beschrieben, die entweder einfach oder zusammen gesetzt sein können. Relationen schließlich beschreiben die Beziehungen zwischen zwei oder mehreren *Features*. Mögliche *Features* in GDF sind Straßenabschnitte und Straßen, Fährverbindungen, Brücken, Kreuzungen oder Eisenbahnen.

GDF zielt damit klar auf reine Navigationsanwendungen. Allerdings konnte gezeigt werden, dass GDF Daten sich in das Umgebungsmodell überführen lassen [VGH+02] und dort dann auch gemeinsam mit Navigations-Daten aus anderen Datenquellen (z.B. ATKIS) verarbeitet werden können.

3.3.6 Physical Markup Language (PML)

Die *Physical Markup Language* (PML) [FAO+03] ist ein Standard, der physische Objekte der Welt für industrielle und kommerzielle Anwendungen beschreiben soll. Durch eine Verbindung von physischen Objekten und

dem Internet sollen Objekte besser verfolgbar sein und auch kontrolliert werden können, z.B. in logistischen Prozessen. Ziel ist es unter anderem, über den elektronischen Produkt-Code EPC, wie er z.B. von an physischen Objekten angebrachten RFID-Tags ausgestrahlt wird, schnell und standardisiert auf die zugehörige Objektinformation über das Internet zugreifen zu können. Dazu wird der EPC-Code über den *Object Name Service* auf eine Internet-Adresse abgebildet, unter der eine PML-Beschreibung des zugehörigen Objekts gefunden werden kann. Damit soll die Vision des *Internet of Things* Wirklichkeit werden. Zur Vermarktung dieser Technologie wurde die Firma *EPCglobal Inc*. [EPC] gegründet.

Als Modellierungsgrundlage für das Umgebungsmodell ist PML ungeeignet. Zwar können zusätzlich zu logistischen Informationen auch Informationen über die physischen Eigenschaften eines Objekts (z.B. gemessene Sensorwerte etc.) modelliert werden. Ortsinformation wird jedoch bevorzugt in einem relativen Koordinatensystem angegeben (z.B. Produkte auf einer Palette, Palette in einem Lastwagen,...). Dies macht es schwierig, von der absoluten Position einer mobilen Anwendung (die z.B. über GPS gewonnen werden kann) auf die Nähe zu anderen Objekten zu schließen. Wie auch in den Kapiteln 4 und 6 zu sehen ist, enthält das Umgebungsmodell weit aus mehr Informationen, als sie über PML spezifiziert sind. Dazu kommt, dass zum Zeitpunkt der Konzeption des Umgebungsmodells noch keine Informationen über PML verfügbar waren. Es sollte jedoch einfach möglich sein, eine Transformation von PML-Objekten in das Umgebungsmodell anzugeben (z.B. über erweiterte Schemata, siehe Kapitel 7.1).

3.3.7 Zusammenfassung

Zusammenfassend kann festgestellt werden, dass kein existierender Geodatenstandard den Anforderungen des Umgebungsmodells vollständig gerecht wird. Zwar könnten bestehende Standards um neue Konzepte erweitert werden, um die Ideen dieser Arbeit umzusetzen. Allerdings würde dies zu einer wesentlich komplexeren Lösung führen, was der Anforderung 13 (Einfachheit des Umgebungsmodells) widersprechen würde. Zum Teil können jedoch Konzepte der Standards übernommen und in die Modellierung integriert werden. Zudem ist es möglich, Transformationen der Standards in das Umgebungsmodell zu entwickeln, um

bestehende Datenbestände im Umgebungsmodell verwenden zu können (was auch die Anforderung 4 (Integration bestehender Informationsdienste) erfordert).

3.4 World Wide Web

Die ersten Ideen für das World Wide Web (WWW) wurden 1989 von Tim Berners-Lee am CERN entwickelt [Ber00]. Das Ziel war es, ein Netz von Daten zu entwickeln, die von jedem Computer aus zugreifbar sind, um die tiefen Gräben zwischen Formaten und Betriebssystemen zu überwinden. Es war zwar bereits möglich, über das Internet auf entfernte Dokumente zuzugreifen, z.B. über das *File Transfer Protokoll (FTP)*, aber dies erforderte immer noch Kenntnis des Aufbaus des anderen Rechners, und es gab kein einheitliches Dateiformat, so dass nur solche Dokumente geladen werden konnten, deren Format vom eigenen Rechner verstanden wurde.

3.4.1 Technische Grundlagen

Die Grundlage des WWW bilden drei Spezifikationen:

- der *Uniform Resource Idenfier (URI)* [URI]
- das *Hypertext Tranportation Protocol (HTTP)* [HTTP] und
- die *Hypertext Markup Language (HTML)* [HTML]

Ein URI ist der Name und zugleich die Adresse eines Objekts im Internet. Er besteht aus einem Schemanamen (z.B. *file, http, ...*) gefolgt von einer Adresse, deren Format vom Schema vorgegeben ist. Im Schema *http* ist dies der Rechnername, gefolgt vom Pfad und dem Dokumentenname, z.B. *http://www.w3c.org/Addressing/URL/uri-spec.html*.

HTTP ist ein Protokoll, mit dem ein Server mit einem Client Dokumente austauschen kann. Die erste Version (HTTP/0.9) diente dem Austausch von Rohdaten über das Internet. HTTP/1.0 wurde durch die Möglichkeit erweitert, Metadaten über die Dokumente mitzulieferen. HTTP/1.1 [HTML] ist die heute eingesetzte und stabile Version des Protokolls, bietet weitere Funktionen und eine stringentere Definition. HTTP ist generisch und zustandslos und basiert auf einem Anfrage/Antwort (*request/response*)-Paradigma.

HTML ist ein generisches Dokumentenformat für Hypertext-Dokumente (die Verweise zu anderen Dokumenten enthalten können). Der Text des Dokuments wird durch sogenannte *tags* ausgezeichnet. HTML basiert auf SGML [SGML], wurde allerdings stark vereinfacht. Ziel von HTML ist es, die Darstellung eines Dokuments von dessen Struktur zu trennen. So spezifiert HTML beispielsweise nur, dass ein bestimmter Text eine Überschrift sein soll, aber nicht, wie (in welcher Größe, Farbe,...) diese auf dem Bildschirm angezeigt werden soll. Diese Trennung wurde in weiteren Spezifikationen von HTML zwar aufgeweicht (z.B. durch **<FONT>**-Angaben), allerdings steht mit *Cascading Style Sheets* [CSS] inzwischen ein eigener Standard für Formatierungsangaben für HTML-Dokumente zur Verfügung.

Das WWW sollte ein weltweites, offenes Hypertextsystem werden. Grundlegend ist dabei, dass es kein System gibt, das eine zentrale, kontrollierende Instanz wäre, sondern nur IDs/Lokatoren (URI), Protokolle (HTTP) und eine Austauschsprache (HTML). Tim Berners-Lee schreibt in [Ber00], dass es zu Beginn sehr schwierig war, diese Vorstellung anderen zu vermitteln. Die Hersteller von Hypertextsystemen beharrten darauf, dass es eine zentrale Link-Datenbank geben müsse, um ungültige Verweise (*broken links)* zu vermeiden.

Wie wir heute sehen, war diese Offenheit der Schlüssel für den enormen Erfolg des WWW. Als Hauptfaktoren für diesen Erfolg können angesehen werden:

- Offene, einfache Standards als gemeinsame *linuga franca:* URI, HTTP und HTML folgen dem *principle of least power*, also dem Prinzip der geringst-möglichen Mächtigkeit. Damit sind sie einfach zu implementieren.
- Minimalistische Standards: Beim Entwurf wurde darauf geachtet, so wenig wie nötig tatsächlich festzuschreiben, um den Datenanbietern die größtmögliche Freiheit zu ermöglichen. Denn globale Standards festzulegen ist um so schwieriger, je komplexer sie sind.
- Niedrige Einstiegshürden für Datenanbieter: Es ist einfach, einen Webserver einzubringen und zu betreiben, denn die Standards werden aufgrund ihrer Einfachheit auf vielen Plattformen unterstützt.

3.4.2 Abgrenzung

Auf den ersten Blick erfüllt das World Wide Web einige der in Kapitel 2 aufgestellten Anforderungen: es ermöglicht über die URI ID-basierten Zugriff (Anforderung 3), erlaubt die Integration bestehender Informationsdienste (Anforderung 4) und die zahlreichen web-basierten Technologien belegen seine Erweiterbarkeit (Anforderung 6). So erlauben z.B. Web Services oder serverseitige Skripte die Verlagerung rechenintensiver Operationen in die Plattform (Anforderung 5). Das WWW ist für seine Anwendungen hoch-skalierbar (Anforderung 7) und durch sein Design offen (Anforderung 8).

Allerdings ist das Web nicht ortsbasiert. Weder erlaubt es den ortsbasierten Zugriff auf noch das ortsbasierte Ablegen von Information (Anforderungen 1 und 2). Das zugrundeliegende Modell von Hypertext-Dokumenten ist nicht mächtig genug, um ortsbezogene Anwendungen in der geplanten Art zu unterstützten (Anforderung 10) und kann zwar prinzipiell auf geeigneten Web-Servern effizient verwaltet werden (Anforderung 11), allerdings nicht effizient abgefragt werden (Anforderung 9), was an der dem WWW zugrundeliegenden Architektur liegt.

Die in dieser Arbeit angestrebte Lösung übernimmt zwar einige der Konzepte, Entwurfsprinzipien und Technologien des WWW (da es schon viele der Anforderungen erfüllt), unterscheidet sich jedoch in wesentlichen Punkten.

In Abbildung 2 ist der grundlegende Unterschied zwischen der Architektur des WWW und der in dieser Arbeit vorgestellten Plattform für ortsbezogene Anwendungen dargestellt. Das WWW unterstützt ID-basierten Zugriff (über die URI). Dadurch können Anwendungen direkt auf die WWW- Server zugreifen. Ist die ID der Anwendung (bzw. der Benutzerin) nicht bekannt, so kann sie eine Anfrage an eine Suchmaschine stellen.

Suchmaschinen benützen sog. Web-Roboter, die Webseiten auf WWW Servern anfragen und die URI sowie den textuellen Inhalt der Seite indiziert in einer Datenbank (im Bild: Link-DB) abspeichern. Damit kann die Suchmaschine eine Anfrage nach Suchwörtern aus dieser beantworten und der Anwendung zurückgegeben. Aufgrund der Größe des WWW indizieren Suchmaschinen nur einen Teil der vorhandenen Webseiten, und die Ergebnisse einer Suchanfragen liefern häufig viele tausend Tref-

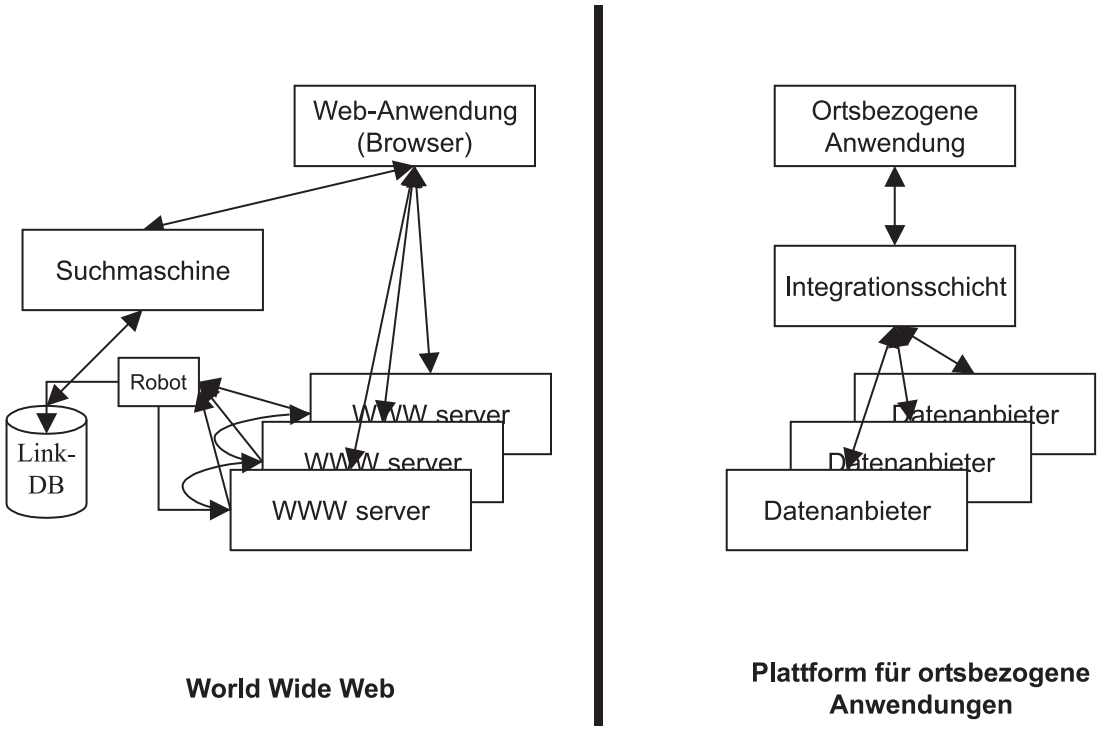

Abbildung 2: Architektur-Vergleich WWW und der angestrebten Plattform

fer. Deswegen wird das Ergebnis nach verschiedenen Kriterien sortiert (*ranking*), in der Hoffnung, dass die URIs von für die Anfrage passenden Dokumenten in der Liste vorne stehen. Die Anwendung kann nun diese URIs verwenden, um auf WWW-Server zuzugreifen und die Dokumente zu holen.

Diese Art des Zugriffs ist für das Umgebungsmodell nicht effizient genug, wie sich in der Analyse von Anforderungen in Kapitel 4 zeigen wird. Um einen möglichst vollständigen Ausschnitt der Umgebung einer Benutzerin bilden zu können, wird in der angestrebten Plattform eine Integrationsschicht vorgesehen, welche die Verteilung der Datenanbieter vor der Anwendung verbirgt und zwischen Anbietern und Anwendung vermittelt. Um für eine große Anzahl von Datenanbietern skalierbar zu bleiben, ist es wichtig, dass die Integrationsschicht eine selektive Vorauswahl unter den Datenanbietern treffen kann. Im WWW wäre dies aufgrund der Heterogenität der auf Webservern angebotenen Daten nicht allgemein möglich. Da die hier angestrebte Lösung jedoch auf die Applikationsdomäne ortsbezogener Anwendungen zielt, kann der Ortsbezug als weltweites Auswahlkriterium genutzt werden, um die Vorselektion

zu treffen. Denn um eine Karte von Stuttgart zu erstellen, müssen nur solche Anbieter gefragt werden, die Daten über das Gebiet von Stuttgart haben.

Zusammenfassend kann man sagen, dass das WWW und die angestrebte Plattform folgende Gemeinsamkeiten haben:

- Entwurfsprinzipien: *principle of least power* und *minimalist principle*. Allerdings wird die vorliegende Lösung aufgrund der erweiterten Anforderungen etwas mächtiger und komplexer werden als das ursprüngliche WWW.
- ID-basierter Zugriff: Auch die vorliegende Lösung wird URIs als IDs und Lokatoren verwenden, allerdings mit einem eigenen Schema
- Protokoll: Als Basis-Protokoll für den Austausch von Umgebungsmodelldaten wird HTTP verwendet.
- *lingua franca*: Für das Umgebungsmodell wird eine Markup-Sprache ähnlich HTML entwickelt, die den plattformunabhängigen Austausch von Daten ermöglicht. Sie ist weniger allgemeingültig als HTML, dafür ist sie semantisch reicher.
- Standards und Erweiterbarkeit: Nach dem minimalistischen Entwurfsprinzip wird so wenig wie nötig in Standards festgeschrieben, die dann für speziellere Zwecke erweitert werden können.
- Nach der Anforderung 4 (Integration bestehender Informationsdienste) wird das WWW als solches in die Plattform integriert.

Grundlegend neu sind dabei folgende Konzepte:

- Das Umgebungsmodell als anwendungsdomänenspezifische Integrationsstrategie: Alle Daten finden im Umgebungsmodell ihren Platz und können darüber in Beziehung gebracht werden.
- Der Ortsbezug als globales Sortierkriterium für die Serverauswahl
- Zum Zugriff auf Daten bei unbekannter ID ist kein *Crawling* nötig (Web-Roboter), sondern die Datenanbieter sind mit Metadaten in einem Verzeichnis registriert.

Das WWW wird ständig weiterentwickelt. Eine zentrale Initiative ist dabei der Versuch, den Dokumenten und Informationen mehr semantisches Wissen zu geben, um effizienter und spezifischer suchen und zugreifen zu können. Diese Entwicklung läuft unter dem Namen *Semantic Web* und nutzt dabei die Erkenntnisse der Ontologie-Techniken (siehe Kapitel 3.2). Die grundlegende Architektur des WWW, wie in

Abbildung 2 skizziert) bleibt davon jedoch weitgehend unberührt. Das Semantic Web wird in Kapitel 3.5 im Vergleich mit anderen Architekturen für die Informationsintegration diskutiert.

3.5 Architekturen für Informationsintegration

Aufgrund der Anforderung 4 (Integration bestehender Informationsdienste) ist es sinnvoll, bestehende Integrationsarchitekturen zu betrachten und zu untersuchen, ob sie als Grundlage für die Systemplattform und die Verwaltung des Umgebungsmodells geeignet sind. Eine häufige allgemeine Architektur solcher Ansätze ist in Abbildung 3 zu sehen. Anwendungen greifen auf eine Integrationsschicht zu, die Anfragen umsetzt und über jeweils angepasste Wrapper an die unterschiedlichen Datenquellen weiterleitet. Die Wrapper sind dafür zuständig, eine Vereinheitlichung der verschiedenen Schnittstellen bereit zu stellen. Die Ansätze unterscheiden sich dabei in verschiedenen Punkten:

- Lokationstransparenz: Müssen die Anwendungen angeben, aus welcher Datenquelle sie Daten erhalten möchten oder gibt es ein allgemeines Modell, das von der Integrationsschicht bereit gestellt wird?
- Dynamik: Kommt das System mit häufig wechselnden Datenquellen zurecht?
- Schema: Welche Art von Datenschema wird der Anwendung zur Verfügung gestellt? Davon wird die Ausdrucksfähigkeit von Anfragen und die Erweiterbarkeit für zukünftige Anwendungen beeinflusst.

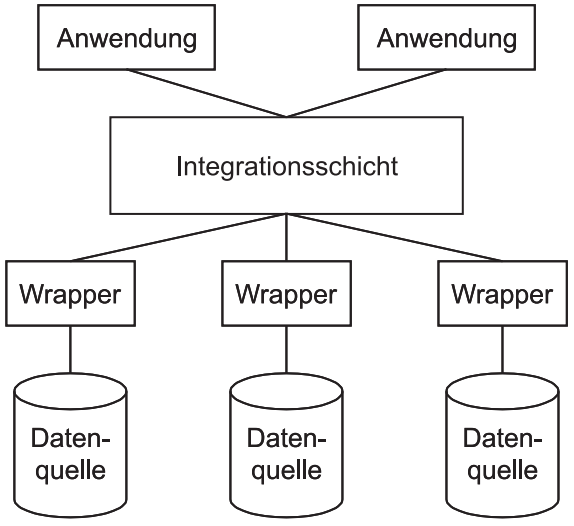

Abbildung 3: Allgemeine Integrations-Architektur

- Integrationsmodus: Welcher Grad an Homogenität wird erreicht?
- Anzahl der Datenquellen (#DQ): Größenordnung der Anzahl der Datenquellen, für welche die Architektur geeignet ist.

In Tabelle 4 ist eine Übersicht über verschiedene Arbeiten und deren Eigenschaften zu sehen, auf die im Folgenden nun kurz eingegangen wird.

3.5.1 EAI Systeme

Kommerzielle Integrationsprodukte (*Enterprise Application Integration*) wie IBMs *Data Joiner* [IBM00], Attunitys *Connect* [Att01] oder *EXIP* [PV02], sowie einige Forschungsprototypen wie TSIMMIS [GPQ97] zielen auf die Integration einer festen Menge von vorab bekannten Datenquellen ab. Auch der Bereich der Schema Integration [BLN86] sowie Software-Systeme für diesen Zweck (z.B. *Clio* [HMN+99]), haben einen ähnlich eingeschränkten Fokus, da sie auf das Problem abzielen, eine feste Menge von Quell-Schemata (und die dazugehörigen Daten) in ein optimal berechnetes oder vordefiniertes Schema zu transformieren. Sie haben deswegen eine eingeschränkte Dynamik, was der Anforderung 8 (Offenheit) widerspricht. Allerdings können solche Systeme benützt werden, um externe Datenquellen in das Umgebungsmodell zu transformieren.

3.5.2 Information Manifold

Bei *Information Manifold* [LRO96] werden verschiedene Datenquellen mittels Joins verbunden. Es werden Anfragen über ein globales Datenschema benützt, um die Inhalte der Datenquellen zu beschreiben, und eindeutige Identifikatoren um Mehrfachrepräsentationen zu verknüpfen. Allerdings werden diese Mehrfachrepräsentationen nicht verschmolzen, und das globale Schema ist nicht erweiterbar. Es werden alle möglichen Kombinationen geeigneter Quellen aufgelistet und das Ergebnis einer Datenquelle als zusätzliche Eingabe für die Anfrage an die nächste Datenquelle verwendet. Betrifft eine Anfrage viele Datenquellen, ist dieses sequentielle Verfahren sehr ineffizient.

3.5.3 Semantische Mediatoren

SEAL [MSS+02] oder *Semantic Mediation* [GLM02] verwenden zur Integration einen Mediator, der die Verteilung der Information auf verschiedene Datenquellen vor der Anwendung verbirgt. Die Wrapper stellen eine Umsetzung zwischen dem Quellschema und einer gemeinsamen Ontologie her. Ontologien enthalten sowohl Schema- wie auch Instanz-Konzepte, allerdings ist das Verfahren nur bei einer beschränkten Menge oder Komplexität der modellierten Daten tauglich. Anfragen werden über logische Deduktion aus der Ontologie heraus beantwortet, was im Allgemeinen für große Datenmengen ineffizient ist.

Tabelle 4: Übersicht über Integrationsarchitekturen (nach [SHG+04])

	Lokations-transparenz	**Dynamik**	**Schema**	**Integrations-Modus**	**#DQ**
EAI Systeme	(ja)	niedrig	global	Schema, *global as view*	<10
Information Manifold	ja	mittel	global	Instanz, *local as view*	$<10^3$
Semantische Mediatoren	ja	mittel	global, aber erweiterbar	Instanz, Konzepte der Ontologie	$<10^2$
Semantic Web	nein	(hoch)	globale Basis, lokale Verfeinerungen	Einheitlicher Zugriff, Auswahl von Konzepten	$>10^9$
GRID	nein	mittel	keines	Einheitlicher Zugriff, Auswahl von Eigenschaften	$<10^3$
Nexus-Plattform	ja	mittel	global, aber erweiterbar	Instanz, *local as view*	$<10^6$

3.5.4 Semantic Web

Das *Semantic Web* ist die nächste Generation des World Wide Web [BF00], bei dem zusätzlich zu menschenlesbaren Web-Dokumenten auch maschinenlesbare Information verfügbar ist. Dadurch können Suchmaschinen wesentlich einfacher die richtigen Informationen zu einer Anfrage herausfinden. Technisch gesehen ist das *Semantic Web* eine Menge von offenen Standards um Ontologien zu repräsentieren (z.B. OWL [OWL], siehe Kapitel 3.2) und Verarbeitungsmethoden.

Das *Semantic Web* beschäftigt sich nur mit Metadaten. Im Gegensatz zu der hier angestrebten Lösung werden keine Daten verarbeitet oder transformiert, um eine Informationsintegration durchzuführen. Seine Technologien können jedoch als Basis dafür verwendet werden, ebenso wie semantische Mediatoren.

3.5.5 GRID

Ein GRID [FK99] ist eine verteilte Systemumgebung, die aus großen und unterschiedlichen Mengen von verteilten Ressourcen und Diensten besteht. Virtualisierungsdienste [RNC+02] werden verwendet, um verschiedene Ebenen der Transparenz zwischen den Anwendungen und den Datenquellen oder Diensten zu schaffen, wie z.B. ein *discovery service*, mit dem Datenquellen nach bestimmten Kriterien ausgewählt werden können. In *Data GRIDs* sind viele kleine und großen Dienste über lose gekoppelte Organisationen verteilt und stellen gemeinsam riesige verteilte Datenmengen bereit [HJK+01]. Der Hauptfokus der Forschung liegt auf der Verwaltung von Repliken und Konsistenzfragen. Die Granularität der Information ist meist auf Dateiebene. Anwendungen können keine komplexen Anfragen schicken, die mehrere Datenquellen überspannen, sondern benutzen das GRID, um genau eine Datenquelle zu finden, welche die gewünschten Informationen komplett liefern kann.

3.5.6 Nexus-Plattform und Abgrenzung

Die Nexus-Plattform ist die in dieser Arbeit vorgestellte Lösung. Aufgrund der in Kapitel 3 aufgestellten Anforderungen ist keiner der hier erwähnten Ansätze vollständig als Lösung tauglich. Viele zielen auf eine vorab bekannte Menge von Datenquellen ab, was der Anforderung 8 (Offenheit) widerspricht. Die Lösung muss Lokationstransparenz bieten, damit die Anwendungen unabhängig von den verfügbaren Datenquellen entwickelt werden können, und auch eine hohe Dynamik bei den Datenquellen unterstützen. Das Umgebungsmodell stellt ein globales Schema dar, aber Anforderung 12 fordert dessen Erweiterbarkeit. Die Integration der Datenquellen erfolgt auf Instanzebene einzelner Objekte. Um der Anforderung 7 (Skalierbarkeit) gerecht zu werden, wurde eine Größenordnung von weniger als 10^6 Datenquellen angenommen: diese liegt zwar drei Größenordnungen unter der des *Semantic Web*, aber es werden auch nicht alle Informationsquellen ortsbezogen sein.

3.6 Plattformen für ortsbezogene Anwendungen

In diesem Abschnitt werden existierende Plattformen untersucht, die ortsbezogene Anwendungen unterstützen. Dabei wird unterschieden zwischen kommerziellen Plattformen, die es ihren Kunden ermöglichen, eigene ortsbezogene Anwendungen zu entwickeln und zu betreiben, und Forschungsarbeiten auf diesem Gebiet.

3.6.1 Forschungsarbeiten

In der Forschung gibt es bisher nur wenige Arbeiten, die auf die Unterstützung verschiedener orts- oder kontextbezogener Anwendungen durch eine Systemplattform zielen.

Henriksen et al. Die Arbeiten von [HI04] kommen der Idee, kontextbezogene Anwendungen über ein Umgebungsmodell zu unterstützen, schon recht nahe. Dort wurde eine graphische Modellierungssprache für Kontextmodelle entwickelt (*context modelling language, CML*) und Werkzeuge, um damit Kontextmodelle für Anwendungen zu erstellen. CML basiert auf dem *Object Role Model* [Hal01] und enthält *Facts* (Daten und Werte, die auch ungenau oder mehrfach vorhanden sein können und Historien besitzen können) und Abhängigkeiten bzw. Beziehungen zwischen *Facts.* Die Kontextmodelle, die mit CML erstellt werden, können dann mit Hilfe von Werkzeugen auf die Infrastruktur bzw. Anwendungen abgebildet werden. Die Infrastruktur von [HI04] besteht aus fünf lose gekoppelten Schichten: Der *context gathering layer* verarbeitet direkte Sensordaten, fusioniert und interpretiert sie, um ein höheres Abstraktionsniveau zu erreichen. Der *context reception layer* bildet den *context gathering layer* bidirektional zum *context management layer* ab, der die Kontextmodelle der Anwendungen verwaltet. Geplant ist eine verteilte Verwaltung; derzeit ist diese Schicht aber noch als einzelne Datenbank realisiert. Im darauf aufsetzenden *query layer* werden Anfragen und Ereignisdefinitionen umgesetzt. Der *application layer* enthält die Anwendungen, die entweder direkt auf den *query layer* oder den dazwischenliegenden *adaptation layer* (für Trigger) zugreifen können.

Auch wenn der Ansatz von [HI04] durch seine Modellbasierung der hier vorgestellten Arbeit ähnelt, gibt es doch signifikante Unterschiede. Anstelle eines umfassenden Umgebungsmodells wird hier für jede neue Anwendung ein neues Kontextmodell angelegt. Zwar besteht die Möglichkeit, Teile der Kontextmodelle zu teilen, aber da dieser Prozess nicht

von vorne herein vorgesehen ist, wird die Konvergenz zwischen den verschiedenen Kontextmodellen recht gering sein. CML unterstützt nur symbolische bzw. relationale Lokationsmodelle. Der Ortsbezug kann nur über Abhängigkeiten zwischen *Facts* modelliert und ausgewertet werden. Wie sich in Kapitel 6.2.7 jedoch zeigen wird, ist für das Umgebungsmodell ein geographisches Lokationsmodell sinnvoll und vorgesehen. Für die Modellverwaltung ist zwar eine verteilte und skalierbare Lösung geplant, diese ist laut [HI04] jedoch noch nicht implementiert. Hier liefert die vorliegende Arbeiten Konzepte, wie im globalen Stil unabhängige Datenanbieter und Anwendungen zusammenarbeiten können. Insgesamt zielt das Projekt darauf ab, in einem organisatorisch abgeschlossenen Szenario die Entwicklung verschiedener kontextbezogener Anwendungen zu unterstützen.

Aura. Im Aura-Projekt [GSS+02] werden Anwendungen erforscht, die über verschiedene Umgebungen hinweg durchgängig arbeiten können. Mit dem Aura Context Information System [JS03] werden dabei Kontextinformationen über Benutzer, Orte, Geräte und Netzwerkkonnektivität verwaltet.

Die Systeminfrastruktur von Aura [JS03] bietet nicht die Skalierbarkeit, die in dieser Arbeit angestrebt wird. Das entstehende Kontextmodell scheint nicht so leicht erweiterbar zu sein, wie es in Anforderung 6 (Erweiterbarkeit bezüglich neuer Anwendungen) gefordert wird.

iQueue. Für [CPW+02] ist das Hauptproblem kontextbezogener Anwendungen die Komposition von Daten unzähliger Datenquellen und Sensoren. Die Autoren nennen sowohl nicht ortsbezogene Szenarien wie Aktienmarktbeobachtung in Kombination mit persönlicher Kontobewegung als auch ortsbezogene Szenarien wie die Anfrage nach freien Taxis oder Personen in einem bestimmten Umkreis. Das *iQueue*-System bietet Anwendungsentwicklern die Möglichkeit, sogenannte *Composer* mittels einer funktionalen Spezifikationssprache *iQL* [CLC+02] zu definieren oder mittels Programmcode und Fragmenten zu programmieren. *Composer* setzen Eingabedaten in Ausgabedaten um und können einen Verarbeitungsbaum bieten. Da sie nicht fest mit der Datenquelle verdrahtet sind, sondern nur die Eigenschaften der Daten spezifiziert wird, ist ein dynamisches Binden und Austauschen der Datenquellen und *Composer* möglich. Die Ausführung von Composern kann sowohl von Anfragen ausgelöst werden (*pull*) als auch von sogenannten *data change events* (*push*,

trigger). Der *Composer-Manager* instanziiert und installiert *Composer* und kann so z.B. durch Lastbalanzierung und Replikation für hohe Skalierbarkeit sorgen. Sobald ein neuer *Composer* installiert wird, versendet er eine Ankündigung mit einer Beschreibung seiner Ausgabe, die es ermöglicht, ihn dynamisch an andere, daran interessierte *Composer* zu binden.

Das *iQueue*-System bezeichnet sich zwar als hochgradig skalierbar und zielt auf umfangreiche Szenarien (mit „*Millionen* von Dateien und Datenquellen, *Milliarden* von mobilen Geräten und *Trillionen* von physischen Sensoren“, nach [CPW+02])). Allerdings hat es ein allgemeines Verarbeitungsmodell, das ortsbezogene Anfragen und Events nicht direkt unterstützt: zwar können innerhalb von *Composern* räumliche Funktionen verwendet werden, diese müssen jedoch „von Hand“ entwickelt werden. Das verwendete Typmodell (von [XML Schema]) enthält keine räumlichen Datentypen. Auch nutzt die Verarbeitung keine räumlichen Algorithmen (wie z.B. Ankündigungen nur in einem räumlich begrenzten Gebiet zu versenden statt weltweit). Außerdem kommt das *iQueue*-System ganz ohne das Konzept eines umfassenden Umgebungsmodells aus. Damit bleibt die tatsächliche Semantik der Daten ein ungelöstes Problem, wie auch im Ausblick von [CPW+02] erwähnt wird.

Keine der untersuchten Forschungsprojekte kann die Anforderungen dieser Arbeit vollständig erfüllen, auch da sie mit anderen Voraussetzungen und Annahmen über die bestehende und zukünftige Systemwelt arbeiten.

3.6.2 Kommerzielle Plattformen

In [HMR02] wurden neun verschiedene kommerziellen Plattformen nach verschiedenen funktionalen und nichtfunktionalen Kriterien untersucht und mit der Nexus-Plattform verglichen. Dabei wurden die Kriterien (in denen sich die Anforderungen aus Kapitel 2 widerspiegeln) mit Gewichten versehen und die vier nach Punkten besten Plattformen einer detaillierteren Analyse unterzogen. Im Folgenden werden nur diese vier „Testsieger“ vorgestellt und gegen die Anforderungen geprüft.

[Cellpoint] besteht aus zwei Hauptprodukten, dem *Mobile Location System (MLS)* und dem *Mobile Location Broker (MLB)*. Während das MLS für die Verwaltung von mobilen Objekten zuständig ist (und dafür verschiedene Lokalisierungssysteme unterstützt), ist der MLB eine Plattform, die zwischen dem MLS und ortsbasierten Anwendungen vermittelt und Authen-

tifizierung, Autorisierung und Datenschutz-Funktionen zur Verfügung stellt. Über den MLB können ortsbezogene Informationen und Dienste bereitgestellt werden.

[Esri] bietet eine Plattform unter dem Namen *ArcLocation* an, die Dienste für Mobilfunkbetreiber und Entwickler ortsbezogener Anwendungen bereitstellt. *ArcLocation* besteht aus einem räumlichen Server mit verschiedenen standardisierten Schnittstellen, dem *Connector*, mit dem verschiedene Dienste miteinander verbunden werden können, einer Integrationsmöglichkeit für Lokalisierungssysteme und einem *Application Server* für mobile Anwendungen. Außerdem stehen Werkzeuge zur Anwendungsentwicklung zur Verfügung.

Der Kern des [Gate5] Produkts ist die *zone5 Engine*, ein Rahmenwerk für die Entwicklung und Verbreitung von ortsbezogenen Anwendungen und Diensten. Acht verschiedene Funktionsblöcke bieten Grundfunktionalität für Anwendungen an: Lokalisierungsdienste, Navigation, Kartendienste, Gruppenkollaboration, Personalisierung, integrierter Nachrichtenaustausch, Zugangsverwaltung und ein einheitlicher Speicher für Inhalte. Ressourcenstärkere Endgeräte können Teile dieser Plattform lokal nutzen, während ressourcenschwache die Komponenten in der Infrastruktur nutzen. Die *gate5* Plattform ist dazu gedacht, für eine Menge von konkreten Anwendungen konfiguriert und angepasst zu werden.

Die [OpenWave] Plattform besteht aus fünf Komponenten: das *Mobile Access Gateway*, mit dem Internetangebote mit drahtlosen Diensten verbunden werden können, dem *Download Manager*, der für das Herunterladen von Inhalten und Applikationen zuständig ist, einem *Provisioning Manager*, der als zentralisierte Lösung Sprach- und Datendienste zur Verfügung stellt, dem *Location Manager*, der die Position eines Benutzers aus anderen Systemen entgegen nimmt und an die Applikation weitergibt und dem *Location Studio*, das zwischen Anwendungen und den Benutzerdaten vermittelt, um Datenschutz, Authentifizierung und Autorisierung sicherzustellen.

Abgrenzung. Aufgrund von Selbstaussagen und Papierstudium (eine praktische Analyse wäre zu aufwändig) ist davon auszugehen, dass kommerzielle Plattformen bereits einige der Anforderungen zumindest teilweise erfüllen. Allerdings gehen diese Produkte von einem anderen Betreibermodell aus, bei dem ein Anbieter eine oder mehrere ortsbezo-

gene Anwendungen entwickeln und vermarkten möchte. Dazu kauft er sich Inhalte ein und verwendet die Plattform, um seine Anwendungen auf diese Inhalte zugreifen zu lassen. Wenn er selbst kein Mobilfunkbetreiber ist, geht er eine Kooperation mit einem solchen ein, um beispielsweise die Positionsdaten der jeweiligen Mobiltelefone verwenden zu können.

Das Datenmodell ist dabei speziell auf die Anwendungen zugeschnitten und ist in der Regel nicht allgemein genug, um zahlreiche verschiedene Anwendungen unterstützen zu können (wie z.B. *OpenLS*, siehe Kapitel 3.3.2). Die Plattformen bieten zwar Möglichkeiten, verschiedene Datenquellen anzuschließen, aber sie bieten keine fortgeschrittenen Integrationstechniken, wie sie in Kapitel 3.5 diskutiert werden. Auch besitzen die Plattformen hauptsächlich funktionale Schnittstellen von serverseitigen Diensten und keine allgemeine Anfrageschnittstelle für das gesamte Datenmodell: viele bieten nur Anfragen über mobile Objekte (Benutzer) an, während sonstige ortsbasierte Daten nur über speziell entwickelte Dienste in der Infrastruktur (z.B. einen Kartendienst) abfragbar sind. Dafür unterstützen sie aktuelle Kommunikationstechnologien wie SMS oder MMS und bieten Komponenten zur Authentifizierung und Autorisierung an, die in dieser Arbeit nicht geplant sind.

Als Fazit ist zu sagen, dass heutige kommerzielle Produkte zwar als Grundlage für eine Plattform verwendet werden könnten. Um jedoch die Anforderung 6 (Erweiterbarkeit bezüglich neuer Anwendungen), die Anforderung 8 (Offenheit) und die Anforderung 10 (Mächtigkeit des Umgebungsmodells) zu erfüllen, müssen neue Konzepte und Verfahren entwickelt werden. Für die prototypische Implementierung dieser Verfahren wurden neue Komponenten entwickelt, da kommerzielle Plattformen zu kostspielig, vermutlich nicht flexibel genug und zum Zeitpunkt der Entwicklung auch noch nicht im heutigen Maße verfügbar waren.

In Zukunft könnte es Computer geben, die weniger als 1,5 Tonnen wiegen. (Fachblatt Popular Mechanics, 1949)

Kapitel 4: Anwendungsdomäne: ortsbezogene Anwendungen

In diesem Kapitel wird die Anwendungsdomäne ortsbezogener Anwendungen definiert und durch verschiedene Szenarien konkretisiert. Aus diesen Szenarien können dann inhaltliche Anforderungen an das Umgebungsmodell abgeleitet werden.

4.1 Anwendungsdomäne

Zunächst sollen generelle Merkmale der Anwendungsdomäne beschrieben werden, danach fünf typische Anwendungsgebiete und deren Merkmale.

4.1.1 Generelle Merkmale

Wie bereits in Kapitel 2.1 erwähnt, zeichnen sich viele ortsbezogene Anwendungen dadurch aus, dass der Benutzer mobil ist. Damit ändert sich sein Kontext: sprich sein Ort, seine Umgebung, seine Aufmerksamkeit und die Aktivitäten, denen er nachgeht. Kontextbezogene Anwendungen passen sich dieser Kontextveränderung an, ortsbezogene Anwendungen sind also eine Unterklasse der kontextbezogenen Anwendungen, deren Anpassung primär vom Ort der Benutzerin abhängt, z.B. durch:

- ortsbezogene Selektion: die Auswahl der präsentierten Daten,
- ortsbezogene Präsentation: die Darstellung der präsentierten Daten,
- ortsbezogene Aktion: die Anwendungslogik,
- ortsbezogene Speicherung: das Ablegen (und auch das Wiederauffinden) von Daten.

Grundsätzlich lassen sich zwei Systemmodelle ortsbezogener Anwendungen unterscheiden: *mobile Anwendungen* begleiten den Benutzer auf einem kleinen Endgerät, das dieser mit sich herumträgt. In *instrumentierten Umgebungen* laufen die Anwendungen in einer Infrastruktur, die über Sensoren und Aktoren mit dem Benutzer interagiert. Beide Systemmodelle können auch gemeinsam auftreten. Allgemein können mit mobilen Anwendungen größere Gebiete abgedeckt werden, da dort die notwendige Infrastruktur von der Benutzerin mitgenommen wird, während instrumentierte Umgebungen aufgrund des hohen Installationsaufwandes eher auf einzelne Räume oder Gebäude beschränkt sind.

Ein weiteres Merkmal ortsbezogener Anwendungen sind beschränkte Ressourcen. Bei mobilen Anwendungen rühren diese von den technischen Einschränkungen des Endgeräts her: damit es transportabel ist, muss es besonders klein und leicht sein, und ist deswegen in der Regel weniger leistungsstark wie ein stationäres System: es bietet weniger Rechenleistung, weniger Speicher und kann nur über drahtlose Netzwerkverbindungen kommunizieren, die in der Regel weniger Bandbreite als drahtgebundene haben und zudem unzuverlässiger sind.

Aber auch bei instrumentierten Umgebungen kommt es vor, dass die Anwendungen unter beschränkten Ressourcen entwickelt werden, da sie häufig für eingebettete Systeme entwickelt werden – wenn Alltagsgegenstände mit Rechenleistung ausgestattet werden, sind komplette Computer nicht wirtschaftlich.

Ortsbezogene Anwendungen benötigen eine vielseitige technische Infrastruktur: Um den Kontext der Benutzerin zu erfassen, sind Sensoren notwendig, die Zustände der physischen Welt (z.B. der Ort des Benutzers oder die Temperatur eines Raumes) erfassen und digital zur Verfügung stellen, sei es lokal am mobilen Endgerät oder in der instrumentierten Umgebung. Sollen Aktionen in der realen Welt ausgelöst werden, so benötigt es Aktoren, sprich Geräte, die von der Anwendung angesteuert werden können (z.B. ein Lichtschalter, eine Heizungssteuerung oder ein Videoprojektor). Zur Kommunikation müssen je nach Verfügbarkeit verschiedene Systeme eingesetzt werden, z.B. WLAN, Infrarot oder UMTS. Diese Vielfalt an externen Systemen geht einher mit einer Vielfalt an unterschiedlichen Protokollen, Schnittstellen und Datenformaten, mit

denen die Geräte angesprochen und ausgelesen werden. Ein Ziel dieser Arbeit ist es daher auch, hier eine Vereinheitlichung zu schaffen, um die Heterogenität der Infrastruktur vor den Anwendungen zu verbergen.

4.1.2 Anwendungsgebiete

In der Domäne ortsbezogener Anwendungen können verschiedene Anwendungsgebiete definiert werden, die jeweils spezielle Merkmale aufweisen. Diese Gebiete überlappen sich, d.h. konkrete Anwendungen können in mehrere Anwendungsgebiete fallen.

Ortsbezogene Informationssysteme. Ortbezogene Informationssysteme sind in der Regel mobile Anwendungen, welche die Benutzerin mit zusätzlichen Informationen zu ihrer jeweiligen Umgebung versorgen, z.B. durch eine Karte oder durch Erklärungen, die nach einer Zeigegeste („was ist das?“) angezeigt oder anderweitig ausgegeben werden. Weit verbreitet sind touristische Szenarien mit Informationen über Sehenswürdigkeiten, Unterkunft und Verpflegung, oder auch universitäre Campus-Informationssysteme, die z.B. Studierende oder Besucher über Fakultäten, zentrale Einrichtungen etc. informieren. Häufige Funktionen dieser Systeme sind die Suche nach nahen Objekten, entweder über ein Gebiet (*Window-Query*) oder sortiert nach der Entfernung (*Nearest Neighbor-Query*) sowie die selbstständige Anzeige von Informationen, wenn der Benutzer einen bestimmten Ort erreicht hat. Zudem ist häufig eine Navigationsfunktion integriert (s.u.).

Die Präsentation basiert meist auf einer Karte, auf der relevante Objekte markiert werden, über die weitere Information abgerufen werden kann. Es gibt auch rein akustische Informationssysteme, bei denen die Benutzerin über einen Kopfhörer informiert wird.

Navigation. Unter Navigation wird die Aufgabe verstanden, den Benutzer auf bestem Weg von einem Start- zu einem Zielort zu leiten bzw. eine optimale Rundtour zwischen gegebenen Punkten zu finden. Die Kriterien für die beste Route können dabei von den Benutzervorlieben bestimmt werden, z.B. die schnellste, die kürzeste oder die mit den wenigsten Treppen. Die Berechnung der Route findet i.d.R. auf einem Verkehrsnetzwerk statt, z.B. Straßen oder dem öffentlichen Nahverkehr. Navigationsanwendungen sind bereits weit verbreitet, besonders im Bereich Fahrzeugnavigation. Von dynamischer Navigation spricht man, wenn die Route unterwegs den aktuellen Gegebenheiten angepasst wird (z.B. Neuberech-

nung, wenn die Benutzerin einen anderen Weg einschlägt oder ein Hindernis auftaucht). Multi-modale Navigation bedeutet, dass verschiedene Verkehrsmittel gewählt und auch zwischen diesen gewechselt werden kann.

Die berechnete Route wird meist mit einer oder mehrerer der folgenden drei Möglichkeiten präsentiert: sie wird in eine Karte eingezeichnet, es werden Pfeile angezeigt, die angeben, in welche Richtung sich der Benutzer bewegen soll, oder es wird eine textuelle Wegbeschreibung gegeben, die dann häufig auch wichtige Wegpunkte zur Orientierung enthält (z.B. „an der Kirche links abbiegen“).

Umgebungsannotation. Solche ortsbezogene Anwendungen ermöglichen es, an einem Ort Nachrichten zu hinterlassen, die dann von anderen Benutzern gelesen werden können. Dabei ist es u.U. auch möglich, den Kreis der Benutzer einzuschränken, welche die Nachricht lesen dürfen. Während die bisherigen Anwendungsgebiete davon ausgehen, dass Anwendungen die Umgebungsinformation nur auslesen, werden hier durch die Anwendung neue Umgebungsinformationen angelegt und gespeichert.

Die Präsentations- (bzw. Interaktions-)form ist hier ein spezieller Editor, mit dem die Nachricht verfasst oder aufgenommen werden kann.

Instrumentierte Umgebungen. In diesem Anwendungsgebiet werden verschiedene Anwendungen zusammengefasst, die eine instrumentierte Umgebung voraussetzen, da der Fokus dieser Arbeit mobile, ortsbezogene Anwendungen sind: man könnte dieses Anwendungsgebiet noch in Büro-, Heim- und Produktionsumgebungen unterteilen. Allgemein unterstützen diese Anwendungen den Benutzer bei seinen Aufgaben, in dem unterschiedliche Gegenstände und Geräte „intelligent“ auf seinen konkreten Kontext (seinen Ort und seine Aktivitäten) reagieren.

Die Präsentation ist dabei stark vom technischen Kontext des Benutzers abhängig, sprich davon, welche Geräte gerade in seiner Nähe verfügbar sind.

Spiele und Edutainment. Dieses Anwendungskonzept umfasst Spiel-, Unterhaltungs- und erlebnisorientierte Lernkonzepte, welche die neuen Möglichkeiten ortsbezogener Anwendungen nutzen. Sie können sowohl als mobile Anwendungen als auch in instrumentierten Umgebungen eingerichtet werden. Gemein ist diesen Anwendungen, dass sie Anreize zur

Tabelle 1: Zuordnung der Szenarien zu den Anwendungsgebieten

Szenario \ Gebiet	Ortsbez. Info.-Sys.	Navigation	Anno-tation	Instrum. Umg.	Spiele
Stadtinfo	•	•			
Erinnerung	•		•	•	
Nachrichten	•				
Navigation		•			
Rallye			•		•
Rollenspiel	•	•	•	•	•
Kommunikation		•		•	
Wartung	•			•	

Benutzung schaffen, wie z.B. Wettbewerb, unterhaltsame Multimedia-Präsentationen oder Preise. Dadurch lässt sich häufig neuartige oder ungewohnte Technologie leichter am Menschen erproben. Oft bringen diese Anwendungen virtuelle Objekte in die reale Umgebung ein, die nur im Spielkontext eine Bedeutung erlangen und auch nur durch die Anwendung selbst sichtbar werden (z.B. Schätze, die gefunden werden können oder virtuelle Personen, die man verfolgt). In diesem Fall wird von *mixed reality* (gemischter Wirklichkeit) gesprochen.

Die Präsentation ist so vielfältig wie die Spielideen, die Entwickler umsetzen. In [NPM01] wird aufgezeigt, wie klassische Spielkonzepte aus Brett-, Computer- oder Rollenspielen auf mobile, ortsbezogene Anwendungen umgesetzt werden können.

4.2 Szenarien

An dieser Stelle werden die oben skizzierten Anwendungsgebiete durch Anwendungsszenarien konkretisiert. In Tabelle 1 wird dargestellt, welche Szenarien welchen Anwendungsgebieten zuzuordnen sind. Die Szenarien können dabei – je nach Realisierung – auch mehreren Anwendungsgebieten zugeordnet sein. Ziel dieser Betrachtung ist es,

inhaltliche Anforderungen an das Umgebungsmodell abzuleiten, sprich festzustellen, welche Informationen und Daten dort enthalten sein müssen.

Mobiles Stadtinformationssystem. Ein mobiles Stadtinformationssystem läuft auf einem kleinen Endgerät (z.B. einem PDA) kann sich entweder über WLAN (so verfügbar) oder über UMTS drahtlos mit dem Internet verbinden. Zudem verfügt es über einen GPS-Sensor, über den im Freien die eigene Position mit einer Genauigkeit von 3-10m festgestellt werden kann. Die Standard-Ansicht auf dem Display ist eine Karte der Umgebung, in der die eigene Position eingezeichnet ist. Über ein Menü können verschiedene Suchanfragen nach nahen Objekten gestellt werden, z.B. nach Hotels, Restaurants, Taxis, Sehenswürdigkeiten, Kirchen oder Informationszentren, die dann in der Karte angezeigt werden. Durch Auswahl eines dieser Objekte wird eine Seite angezeigt, auf der in Kurzform die wichtigsten Informationen (Kurzbeschreibung, Öffnungszeiten, etc.) vermerkt sind sowie ein Verweis auf Webseiten. Außerdem kann sich der Benutzer zu einem dieser Objekte oder zu einer beliebigen Adresse navigieren lassen. Die Daten, welche die Anwendung benötigt, befinden sich teilweise im lokalen Speicher und werden bei Bedarf über die Netzwerkverbindung nachgeladen (siehe auch Szenario Optimierte Kommunikation). Ein Beispiel für ein solches Stadtinformationssystem ist der *NexusScout (*Kapitel 8.3 und [NGS03]).

Ortsbezogener Erinnerungsdienst. Mit einem ortsbezogenen Erinnerungsdienst kann sich die Benutzerin an etwas erinnern lassen, sobald sie einen bestimmten Ort betritt oder verlässt. Er besteht aus mehreren Komponenten: eine Web-Anwendung, mit der sie von jedem Browser aus komfortabel über eine Karte oder eine personalisierte Liste von wichtigen Orten spezifizieren kann, welche Nachricht sie bei welcher Ortsveränderung erhalten möchte; einem Erinnerer, der auf ihrem mobilen Telefon läuft und mit einem speziellen Klingelton auf sich aufmerksam macht und die Nachricht anzeigt; und verschiedenen Infrastrukturkomponenten (z.B. wie in [BR04]), die über Sensoren ihre Ortsveränderungen beobachten und bei Bedarf den Erinnerer informieren. Je nach Detailgrad dessen, was beobachtet werden soll, ist ein Erinnerungsdienst evtl. nur in einer instrumentierten Umgebung sinnvoll, in der viele Sensoren für die Beobachtung zur Verfügung stehen.

Ortsbezogene Nachrichten. Bei einer ortsbezogenen Nachricht (siehe Geocast [DR03]) wird der Empfänger nicht durch einen eindeutigen Namen (z.B. seine E-Mail-Adresse), sondern durch ein geographisches Gebiet spezifiziert. Alle qualifizierten Empfänger, die sich in diesem Gebiet aufhalten, erhalten die Nachricht. Die Benutzerschnittstelle besteht aus einem Editor zum Verfassen der Nachricht und einer Möglichkeit, das Empfänger-Gebiet zu spezifizieren: entweder wird ein räumlich ausgedehntes Objekt aus einer Liste ausgewählt (z.B. „Informatik-Gebäude“, symbolische Adressierung) oder es wird auf einer Karte ein Gebiet angegeben (geographische Adressierung).

Multimodale, dynamische Navigation. Diese fortschrittliche Navigationsanwendung berücksichtigt die persönlichen Vorlieben und Möglichkeiten seiner Benutzerin und findet den optimalen Weg. So werden die verschiedenen Verkehrsmöglichkeiten (Fußmarsch, öffentlicher Nah- und Fernverkehr, *Car-Sharing*-Angebote, *Park&Ride*-Parkplätze, etc) miteinander kombiniert und unterwegs ständig aktualisiert (z.B. bei Staus oder gesperrten Streckenabschnitten). Die Präsentation erfolgt entweder durch eine Karte oder durch textuelle Richtungsangaben.

Ortsbezogene Rallye. Eine Rallye ist ein Spiel, bei dem die Spieler meist unter Zeitdruck bestimmte Aufgaben lösen müssen, die erfordern, dass sie sich dabei bewegen. Wird so ein Spiel als ortsbezogene Anwendung realisiert [NHM+04], so erhalten die Spieler mobile Endgeräte, mit denen sie die Aufgaben angezeigt bekommen und die Antworten eingeben. Zudem können manche Aufgaben als gelöst gezählt werden, sobald der Spieler einen bestimmten Ort erreicht bzw. gefunden hat. Eine Karte wird nicht angezeigt, stattdessen ist es ein Ziel des Spiels, dass die Spieler das Gebiet, in dem die Rallye stattfindet, durch die Aufgaben besser kennenlernen. Wenn es möglich ist, dass Spieler anderen Spielern Nachrichten oder Hinweise hinterlassen, so fällt dieses Szenario auch in das Anwendungsgebiet Annotation. In Kapitel 8.5 wird eine solche Anwendung (die *NexusRallye)* detaillierter beschrieben.

Ortsbezogenes Rollenspiel. Ein ortsbezogenes Rollenspiel (z.B. [Fre03]) ermöglicht es dem Spieler, neben der normalen Alltagswelt noch eine zweite Wirklichkeit in seiner Phantasie zu erleben. Diese Spielwelt entsteht durch die Interaktion mit der Anwendung und mit anderen Spielern. In der Spielwelt führt der Spieler einen Charakter, d.h. eine fiktive Figur, deren Handlungen er bestimmt. Dieser Charakter hat bestimmte

Eigenschaften (z.B. Stärke), die seine Handlungsmöglichkeiten in der Spielwelt einschränken oder erweitern. Die Spielwelt ist mit der Alltagswelt überlagert: bestimmte Orte der Alltagswelt (z.B. eine Kirche) können in der Spielwelt eine besondere Bedeutung haben (z.B. Heilung des Charakters). Dazu kann es virtuelle Spielobjekte geben, die an bestimmten Orten aufgefunden werden können: sprich, der Spieler kann sie durch seine Anwendung sehen, sobald er sich an diesem Ort aufhält. Ein ortsbezogenes Rollenspiel kann prinzipiell rund um die Uhr gespielt werden. Sobald der Spieler an einen relevanten Ort der Spielwelt kommt oder einen anderen Spieler trifft, macht ihn die Anwendung darauf aufmerksam und er kann sich entscheiden, ob er eine Spielhandlung vornimmt oder nicht. Alternativ kann er auch jeden Ort zu einem spielrelevanten Ort machen, in dem er dort Nachrichten für andere Spieler hinterlässt. In Kapitel 8.4 wird eine solche Anwendung detaillierter beschrieben.

Optimierte Kommunikation. Optimierte Kommunikation ist ein Systemdienst, der es mobilen Anwendungen ermöglicht, jederzeit den besten (günstigsten oder schnellsten) Kommunikationskanal (z.B. WLAN wenn verfügbar) zu benutzen. Zudem kann er z.B. durch statistische Daten (*Hoarding*, [BMR04]) oder durch Kenntnis des Benutzerweges vorausplanen, welche Daten die Anwendungen in Zukunft vermutlich benötigen werden, um diese dann zu laden, solange noch eine schnelle Kommunikationsverbindung besteht.

Wartung instrumentierter Umgebungen. Wie bereits erwähnt, gibt es in einer instrumentierten Umgebung eine Vielzahl unterschiedlicher technischer Systeme, die einen nicht zu unterschätzenden Wartungsaufwand bedeuten, z.B. Kalibrierung oder Austausch von Batterien. Mit dieser Anwendung kann ein Wartungstechniker den Ort und den Zustand der verschiedenen Systeme anfragen, um sie bei Bedarf schnell aufzufinden. Über die Anwendung kann dann einfach auf Bedienungs- und Reparaturanleitungen zugegriffen werden.

4.3 Informationen im Umgebungsmodell

Aus den Szenarien kann nun abgeleitet werden, welche Informationen im Umgebungsmodell enthalten sein sollen. Dabei lassen sich grob vier Gruppen unterscheiden:

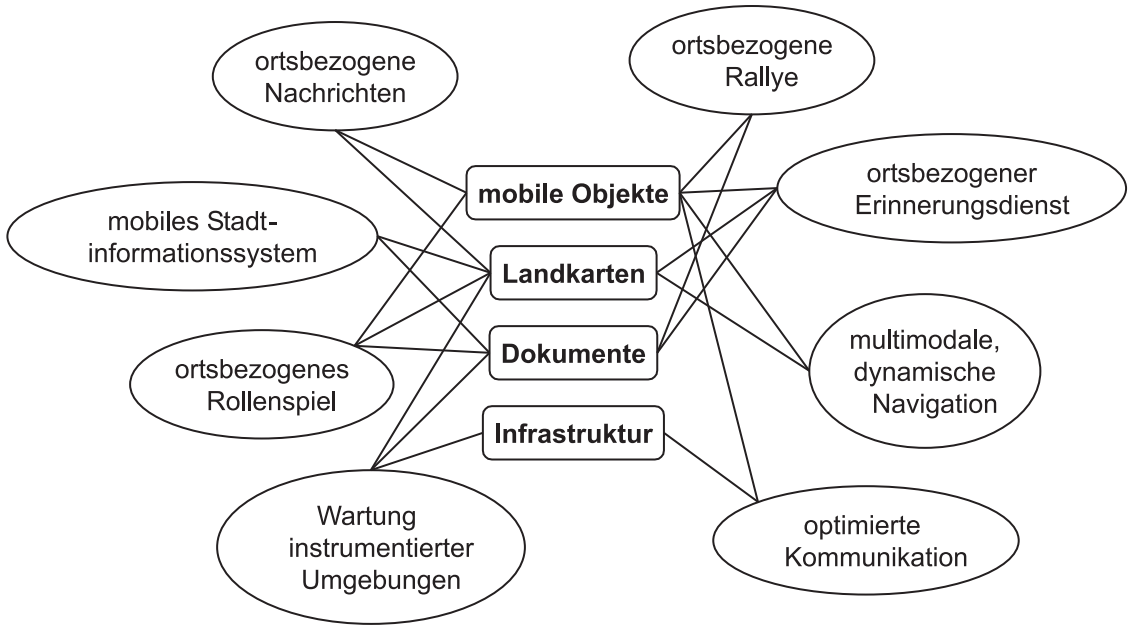

Abbildung 4: Anwendungsfälle und benötigte Daten

- Landkarten: alle Informationen, die schon in herkömmlichen Landkarten einzeichnet sind, sprich Gebäude, Straßen, wichtige Punkte, Räume, Parkplätze, etc.
- mobile Objekte: Personen, Fahrzeuge, Haustiere, etc.
- Infrastruktur: technische Geräte und Informationen über die verfügbare Infrastruktur: Sensoren, Geräte, Verfügbarkeit drahtloser Netzwerke, etc.
- digitale Dokumente: virtuelle Objekte, die nur in der digitalen Wirklichkeit vorkommen und die über einen Ortsbezug mit der physischen Welt verknüpft werden: Notizen, Webseiten, Spielinformationen,...

Abbildung 4 gibt einen Überblick, welche Art von Daten von welchen Anwendungsfällen benötigt werden. Das mobile Stadtinformationssystem zeigt mobile Objekte an (z.B. Taxis), Landkarten und mit wichtigen Objekten verknüpfte digitale Dokumente. Auch das ortsbezogene Rollenspiel benötigt diese Informationen, um die Spielwelt auf die Wirklichkeit abzubilden. Für die Wartung instrumentierter Umgebungen sind in erster Linie Informationen über die Infrastruktur notwendig. Zum Auffinden der Systeme werden jedoch Innenraumpläne benötigt (Landkarten), und Reparatur- oder Bedienungsanleitungen sind Dokumente, die dann mit den jeweiligen Systemen verknüpft werden können. Auch die opti-

mierte Kommunikation benötigt Infrastrukturinformation, insbesondere über die Abdeckung und Zugangspunkte drahtloser Netzwerke. Zusätzlich werden jedoch auch Daten über mobile Objekte, nämlich über die Bewegung der Benutzer verwendet, um gute Abschätzungen über das zukünftige Kommunikationsverhalten treffen zu können. Die vorgestellte Navigationsanwendung verwendet Landkarteninformation (Verkehrsnetze,...), um für die mobilen Objekte (den Benutzer selbst) eine Route zu finden. Aus der Position und Geschwindigkeit anderer mobiler Objekte (z.B. Taxis) können auch Ableitungen über die Verkehrssituation getroffen und beispielsweise ein Stau erkannt werden. Der ortsbezogene Erinnungsdienst beobachtet mobile Objekte, um bei bestimmten Ortswechseln digitale Dokumente anzuzeigen. Auch Landkarteninformation wird benötigt, sofern der Benutzer nur symbolische Gebiete adressiert hat. Ähnliches gilt für die ortsbezogene Kommunikation, wobei hier beim Versenden reiner Textnachrichten keine digitalen Dokumente benötigt werden. Die ortsbezogene Rallye bindet dagegen digitale Dokumente (Aufgabenbeschreibungen) an Orte und benötigt Informationen über mobile Objekte (die Spieler), um festzustellen, wann sie neue Aufgaben erhalten oder bestehende gelöst sind.

Bei dieser Betrachtung fällt auf, dass es einen hohen Grad an Überlappung zwischen den benötigten Daten in den verschiedenen Szenarien gibt. Dies zeigt deutlich, dass es sinnvoll ist, solche Informationen nicht für jede Anwendung extra zu verwalten, sondern ein umfassendes Umgebungsmodell zu entwickeln, auf das verschiedene ortsbezogene Anwendungen zugreifen können, und dieses in einer offenen Plattform zu verwalten.

Im folgenden Kapitel wird nun die Systemarchitektur einer offenen Plattform vorgestellt, die Umgebungsmodelldaten effizient verwalten kann. Anschließend wird Aufbau und Inhalt dieses Umgebungsmodells in Kapitel 6 beschrieben.

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies. (C.A.R. Hoare)

Kapitel 5: Die System-Architektur

In diesem Kapitel wird die System-Architektur einer Plattform vorgestellt, mit der das Umgebungsmodell verwaltet werden soll. Sie wird im Folgenden Nexus-Plattform genannt. Dabei werden einige grundlegende Eigenschaften des Modells angenommen, die erst in Kapitel 6 detailliert erläutert und begründet werden. Diese werden in Kapitel 5.1 kurz vorgestellt. Nach der Beschreibung der Plattformkomponenten (Kapitel 5.2) werden typische Abläufe und Datenflüsse dargestellt (Kapitel 5.3).

5.1 Annahmen über das Modell

Die System-Architektur (im Folgenden Nexus-Plattform genannt) soll das Umgebungsmodell effizient verwalten (Anforderung 11). Um den Aufbau und die Abläufe der Nexus-Plattform darstellen zu können, werden kurz grundlegende Annahmen über das Modell dargestellt, die erst in Kapitel 6 detailliert erläutert und begründet werden.

Das Umgebungsmodell ist *objektbasiert*, es besteht also aus einzelnen Objekten, die sowohl reale Welt als auch virtuelle Information beschreiben (z.B. Häuser, Straßen, Personen oder ortsbezogene Webseiten).

Die Struktur und Semantik des Umgebungsmodells wird durch ein *Datenschema* festgelegt, das Objekttypen definiert: diese legen fest, welche Attribute ein Objekt des jeweiligen Objekttyps haben kann. Der Objekttyp kann wie ein Attribut abgefragt werden.

Zwischen den Objekttypen besteht eine *Vererbungsbeziehung:* erbt ein Objekttyp (Kind) von einem anderen (Vater), so bedeutet dies, dass das Kind alle Attribute des Vaters enthält und zusätzliche Attribute definieren kann.

Es gibt zwei Arten von Datenschemata: das *Standard-Schema* enthält Objekttypen, die allen Komponenten der Nexus-Plattform bekannt sind. *Erweiterte Schemata* enthalten Objekttypen, die nur manchen Datenanbietern und Anwendungen bekannt sind. Die Objekttypen von erweiterten Schemata erben von Objekttypen im Standard-Schema.

Das Umgebungsmodell besteht aus *lokalen Umgebungsmodellen,* die jeweils komplett auf einem Umgebungsmodellserver gespeichert sind. Die lokalen Umgebungsmodelle können sich sowohl räumlich als auch inhaltlich überlappen, so dass das selbe Ding der realen Welt (z.B. der Stuttgarter Fernsehturm) als Objekt in mehreren lokalen Umgebungsmodellen repräsentiert ist.

Alle Objekte des Umgebungsmodells haben eine *eindeutige ID*, genannt NOL (Nexus Object Locator). Diese besteht aus einer Angabe, welcher Server das Objekt speichert, sowie einer weltweit eindeutigen Objekt-ID.

Außerdem haben fast alle Objekte einen *Ortsbezug*, d.h. mindestens eine Position (*pos,* ein geographischer Punkt*)* und eine räumliche Ausdehnung (*extent*, ein geographisches Gebiet oder ein Punkt).

5.2 Komponenten der Nexus-Plattform

In Abbildung 5 ist die Gesamtarchitektur der Nexus-Plattform zu sehen. Sie besteht aus drei Ebenen. In der Dienstebene befinden sich verschiedene *Umgebungsmodellserver*, welche lokale Umgebungsmodelle verwalten.

Sie registrieren Metadaten über ihre Modelle im *räumlichen Verzeichnisdienst* der Föderationsebene. Damit kann die *Föderation* bei einer Anfrage ermitteln, welche Umgebungsmodellserver eine Anfrage überhaupt beantworten können.

Der räumliche Verzeichnisdienst ist eine logisch zentrale Komponente, die jedoch für eine bessere Skalierbarkeit verteilt realisiert ist. Die Föderation ist zustandslos und kann deswegen leicht repliziert werden.

Die Nexus-Plattform kann von mehreren Instanzen verschiedener ortsbezogenen *Anwendungen* genutzt werden (A, B,..., N). Diese greifen in der Regel nicht direkt auf Umgebungsmodellserver zu, sondern richten ihre Anfragen an die Föderation, welche die Anfragen dann an geeignete Umgebungsmodellserver weiterleitet und die Antworten zu einem ein-

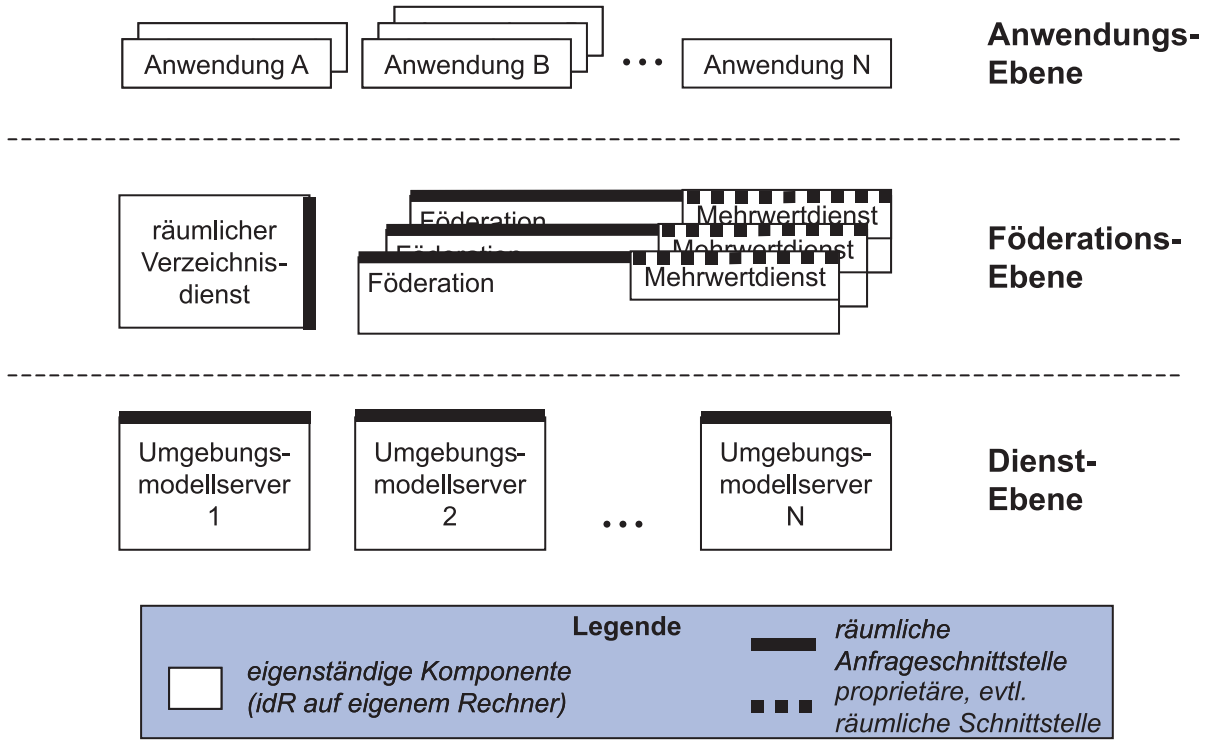

Abbildung 5: Architekturüberblick der Nexus-Plattform

heitlichen Ergebnis integriert. Für Funktionen, die über einfaches Anfragen von Umgebungsmodelldaten hinausgehen, stehen *Mehrwertdienste* zur Verfügung, die in der Föderationsschicht erweiterte Aufgaben übernehmen und den Anwendungen gesonderte Schnittstellen zur Verfügung stellen.

Die Komponenten der Nexus-Plattform werden nun ausführlicher beschrieben.

5.2.1 Räumliche Umgebungsmodellserver

Ein räumlicher Umgebungsmodellserver verwaltet ein oder mehrere lokale Umgebungsmodelle. Er ist definiert durch eine räumliche Schnittstelle, die folgende Funktionen bietet:

- Gebietsanfrage: Objekte, die innerhalb eines gegebenen Gebiets liegen (*within*) oder sich mit diesem überlappen (*overlap*),
- Objektanfrage: Objekte, deren Identität durch eine NOL gegeben ist,
- Nachbarschaftsanfrage: die *n* nächsten Objekte zu einem gegebenen Punkt,

- weitere Einschränkungen: Objekte, deren Attribute gleich oder ungleich einem bestimmten Wert sind,
- boolesche Verknüpfungen davon.

Für diese Anforderungen wurde die Anfragesprache *Augmented World Query Language* entwickelt, die in Kapitel 6.9 näher beschrieben wird.

Der Umgebungsmodellserver registriert jedes seiner lokalen Umgebungsmodelle beim räumlichen Verzeichnisdienst, damit die Föderation entscheiden kann, ob er eine bestimmte Anfrage grundsätzlich beantworten kann oder nicht. Dabei gibt er folgende Metadaten an:

- das Dienstgebiet: ein Polygon, das die räumlichen Attribute aller Objekte des lokalen Umgebungsmodells vollständig umschließt,
- eine Liste von Objekttypen, die er bereits speichert, sowie solche, die er prinzipiell bereit wäre zu speichern.

Da ein räumlicher Umgebungsmodellserver nur über die Schnittstelle definiert ist, sind verschiedene Implementierungen möglich, die jeweils für bestimmte Objekttypen und ihre Eigenschaften optimiert sind, wie z.B.:

- eine Datenbank mit räumlicher Erweiterung, bei der ein Wrapper die Umsetzung der Anfragesprache auf SQL99 [SQL99] vornimmt,
- eine verteilte Hauptspeicherlösung für die effiziente Verwaltung mobiler Objekte, die das Dienstgebiet verlassen können und dann an andere Umgebungsmodellserver weitergegeben werden (siehe auch [Leo03]),
- eine Hauptspeicherlösung für kleine Objektmengen, bei der die Objekte direkt im Speicher gehalten werden ([Grz02] und Kapitel 7.2.1),
- die Anbindung eines bestehenden Systems über einen Wrapper, über den sowohl alte Anwendungen wie auch die Nexus-Plattform auf die bestehenden Daten zugreifen kann ([LBB+04] und Kapitel 7.2.1),
- eine hardwarenahe Lösung mit einem eingebetteten System, bei dem Sensorwerte direkt über die Nexus-Schnittstelle abgefragt werden können ([BBH+03]).

Eine Charakterisierung der Eigenschaften verschiedener Umgebungsdaten und eine Übersicht über unterschiedliche Implementierungen von Umgebungsmodellservern findet sich in [GBH+05].

5.2.2 Räumlicher Verzeichnisdienst (ASR)

Der räumliche Verzeichnisdienst (genannt *Area Service Register,* kurz *ASR)* ermöglicht es, zu einer gegebenen Anfrage zu entscheiden, welche Umgebungsmodellserver sie beantworten könnten (und welche nicht). Dazu werden die Metadaten über die lokalen Umgebungsmodelle ausgewertet. Überschneidet sich das Anfragegebiet nicht mit dem Dienstgebiet eines lokalen Modells, so kann der Umgebungsmodellserver keine Resultate liefern. Werden in der Anfrage Objekte eines bestimmten Typs gefragt, so kommen nur die Umgebungsmodellserver in Frage, die auch Objekte dieses Typs speichern.

Aufgrund dieser Information ist eine Vorauswahl der Umgebungsmodellserver möglich. Das ASR realisiert also einen Index über den Ort und die Objekttypen und verweist auf Umgebungsmodellserver. Ob diese nun tatsächlich Objekte zurückliefern können, hängt allerdings von der aktuellen Datenlage ab. Angenommen, das Datenschema definiert einen Objekttyp „Restaurant“ mit einem Attribut „Nationalität“. Wenn z.B. nach italienischen Restaurants im Umkreis von 100m gefragt wird, so kann das ASR nur ermitteln, welche Server in diesem Gebiet überhaupt Restaurants besitzen. Ob diese italienisch sind, können nur die Umgebungsmodellserver selbst herausfinden.

5.2.3 Föderation

Die Föderation ist für die Zusammenführung der lokalen Umgebungsmodelle zu einem umfassenden Umgebungsmodell verantwortlich. Dazu analysiert sie Anfragen der Anwendungen, stellt über das ASR fest, welche Umgebungsmodellserver sie beantworten könnten, und leitet sie (gegebenenfalls modifiziert) an die Umgebungsmodellserver weiter. Sie wartet auf die Ergebnisse, wobei ein *Time-out* dafür sorgt, dass ein nicht erreichbarer Umgebungsmodellserver nicht die Antwortzeit für die Anwendungen beliebig verlängert. Die Ergebnisse werden nun zu einem einheitlichen Ergebnis zusammengefügt. Dabei kann die Föderation weitere Funktionen durchführen, um das Ergebnis zu verbessern:

- Erkennung von Mehrfachrepräsentationen: Wenn das gleiche Realweltobjekt in verschiedenen lokalen Umgebungsmodellen enthalten ist, kann die Föderation dies über eine Analyse der Attribute feststellen und die beiden Objekte zu einem zusammenfügen.

- Verschmelzen von Mehrfachrepräsentationen: Wenn mehrfachrepräsentierte Objekte erkannt wurden, kann die Föderation versuchen, Eigenschaften von ihnen zu einem einzigen Wert zu verschmelzen. Aus zwei Temperaturwerten kann so z.B. der Mittelwert gebildet werden. Dazu ist Anwendungswissen bzw. Semantik notwendig, diese Aufgabe kann nicht im Allgemeinen Fall gelöst werden. Nur durch zusätzliches Wissen wie z.B. Erhebungszeitpunkt oder Genauigkeit kann die Föderation entscheiden, ob und wie eine Verschmelzung durchzuführen ist.
- Aggregation von geographischen Objekten: Wenn die Anwendung nur an Daten in einer sehr groben Detaillierung interessiert ist, kann die Föderation geographische Objekte aggregieren, z.B. Häuser zu einem Block zusammenfassen. Auch hierfür sind anwendungsspezifische Algorithmen notwendig.
- Schema-Anpassung: Wenn eine Anwendung an Objekten eines Typs interessiert ist, die Föderation von den Umgebungsmodellservern jedoch Objekte eines Untertypen erhält, so kann sie durch Weglassen der zusätzlichen Attribute die Objekte in Objekte des (gewünschten) Obertyps umwandeln.

5.2.4 Mehrwertdienste

Mehrwertdienste sind Anwendungen, die für andere Anwendungen Funktionen anbieten. Sie laufen in der Föderationsebene und haben eine proprietäre Schnittstelle. Allerdings könnte es auch hier Standard-Schnittstellendefinitionen für häufig benötigte Funktionen geben, zu denen dann unterschiedliche Implementierungen existieren.

Im Unterschied zu Anwendungen der Anwendungsebene können Mehrwertdienste direkt auf das föderierte Umgebungsmodell zugreifen, da zwischen der Föderation und einem Mehrwertdienst keine Rechnergrenze existiert. Wie die Analyse gezeigt hat, laufen Anwendungen häufig auf ressourcenbeschränkten Endgeräten. Außerdem gibt es Funktionen, die von verschiedenen Anwendungen in gleicher Art benötigt werden. Mit Hilfe von Mehrwertdiensten kann die Funktionalität der Nexus-Plattform über die reine Abfrage von Umgebungsmodelldaten hinaus erweitert werden. Dadurch können rechenintensive oder häufig benötigte Funktionen von der Anwendungsebene in die Föderationsebene verlagert werden, um dort effizienter ausgeführt zu werden (siehe Anforderung 5).

Für die Nexus-Plattform wurden zwei Mehrwertdienste implementiert, die nun kurz vorgestellt werden.

Navigationsdienst. Navigation ist eine sehr verbreitete ortsbezogene Anwendung. Die Aufgabe besteht darin, für einen Benutzer von einem gegebenen Startpunkt den kürzesten (oder schnellsten) Weg zu einem gegebenen Zielpunkt zu finden (evtl. über gegebene Zwischenpunkte). Dabei muss die Mobilität des Benutzers berücksichtigt werden: gültig sind nur Wege, die der Benutzer auch verwenden möchte und kann. Entweder wird die Suche auf eine bestimmte Mobilitätsart beschränkt (z.B. zu Fuß, mit dem Auto, mit dem Zug). In dem Fall wird der Weg auf einem zur Mobilitätsart passenden Netzwerk gesucht. Oder es wird eine Menge von Mobilitätsarten verwendet, zwischen denen Übergangspunkte bestehen (z.B. ein *Park&Ride*-Parkplatz). In diesem Fall spricht man von multimodaler Navigation. Wird die Navigation als Mehrwertdienst in der Föderation realisiert, so kann sie auf dem föderierten Umgebungsmodell stattfinden, es können also verschiedene Netzwerke für die Routenberechnung kombiniert werden.

Als Schnittstelle für den Navigationsdienst wurde eine XML-Sprache entwickelt, mit der die Anwendung die Parameter für ihre Navigationsanfrage angeben kann (Start-, Ziel- und Zwischenpunkte, Mobilitätsart, etc.), die *Navigation Parameter Language (NPL).* Das Ergebnis einer Navigationsanfrage ist eine Liste von Wegpunkten, die in der *Navigation Result Language (NRL)* zurückgegeben wird [BDG+04].

Kartendienst. Eine weitere häufig verwendete und rechenintensive Funktion ist das Zeichnen einer Karte, in der Umgebungsmodelldaten visualisiert werden (z.B. Häuser, Straßen mit ihrer Ausdehnung und Restaurants mit einem speziellen Symbol). Hierfür wurde ein Kartendienst entwickelt.

Für das Zeichnen einer Karte benötigt der Kartendienst zum einen die Umgebungsmodelldaten, die eingezeichnet werden sollen, zum anderen eine Spezifikation des Bilds, das er zeichnen soll (Bildformat, Auflösung, Farben, Größe, etc.). Eine Anwendung kann dem Kartendienst entweder die Umgebungsmodelldaten direkt übergeben, oder sie schickt ihm eine Anfrage, die er zunächst mit Hilfe der Föderation beantwortet, um dann das Ergebnis einzuzeichnen. Für die Bildspezifikation wurde eine XML-Sprache entwickelt, die *Map Predicate Language (MapPL)* [BDG+04].

5.2.5 Anwendungen

Nexus-Anwendungen sind das Fenster, durch das Benutzer das Umgebungsmodell wahrnehmen und mit ihm interagieren können.

Grundsätzlich existieren zwei unterschiedliche Systemmodelle:

- Mobile Anwendungen laufen auf einem mobilen Endgerät, das die Benutzerin mit sich herumträgt. Meist verfügt es über die Möglichkeit, drahtlos zu kommunizieren sowie über einen oder mehrere Sensoren, um den Ort und andere Umgebungsparameter erfassen zu können. Die Mobilität erfordert möglichst leichte und kleine Geräte, die deswegen in ihrer Leistungsfähigkeit und der Benutzungsschnittstelle eingeschränkt sind.
- Infrastruktur-basierte Anwendungen laufen auf nicht-mobilen Systemen. Sie können über verschiedene Sensoren den Benutzer und seine Umgebung wahrnehmen und verwenden geeignete Geräte in seiner Nähe, um mit ihm zu interagieren. Solche Anwendungen sind typisch für instrumentierte Umgebungen und erfordern einen erhöhten Installationsaufwand. Deswegen ist ihre Reichweite meist auf einen oder wenige Räume begrenzt.

Nexus-Anwendungen benötigen Daten aus dem Umgebungsmodell, um ihre Funktion erbringen zu können. Auch hierfür gibt es verschiedene Möglichkeiten, die auch gemeinsam eingesetzt werden können:

- Sie nutzen die Anfrageschnittstelle der Föderation um sich jeweils die benötigten Daten herunter zu laden, Daten einzufügen oder zu ändern.
- Sie definieren Prädikate auf das Umgebungsmodell, die bestimmte Ereignisse beschreiben. Tritt ein solches Ereignis ein, so erhalten sie eine Benachrichtigung. Die Realisierung eines solchen Ereignisdienstes ist nicht Teil der vorliegenden Arbeit; für eine mögliche Konzeption können die Arbeiten von Martin Bauer [BR04] eingesetzt werden.
- Sie nutzen Mehrwertdienste, die ihnen eine erweiterte Sicht auf das Umgebungsmodell ermöglicht (s.o.)
- Sie nutzen einen eigenen Server, auf dem spezielle Daten für diese Anwendung gespeichert sind.

Weitere Details zum Aufbau und zur Entwicklung von Nexus-Anwendungen finden sich in Kapitel 8.

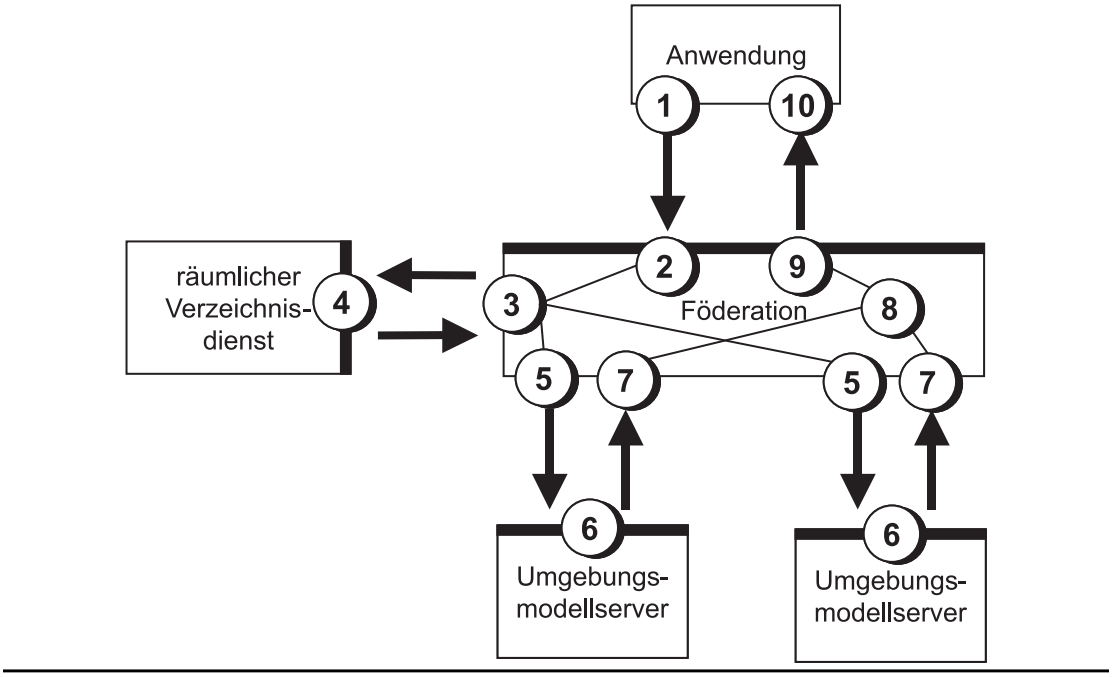

Abbildung 6: Bearbeitung einer Anfrage in der Nexus-Plattform

5.3 Abläufe in der Nexus-Plattform

Um die Funktionsweise der Nexus-Plattform zu verdeutlichen, werden nun verschiedene übliche Abläufe schrittweise erläutert: zunächst werden Anfrage- und Manipulationen des Umgebungsmodells über die Föderation vorgestellt, danach beispielhaft zwei Mehrwertdienste. Abschließend erfolgen einige Bemerkungen zur Optimierung dieser Abläufe.

5.3.1 Anfrage einer Anwendung an die Föderation

Abbildung 6 zeigt die Bearbeitung einer Anfrage in der Nexus-Plattform.

1. Die Anwendung schickt eine Anfrage an die Föderation.
2. Die Föderation analysiert diese Anfrage:
   - Enthält die Anfrage räumliche Beschränkungen? Dies kann z.B. ein Gebiet sein, das die angefragten Objekte umschließt oder schneidet. In diesem Fall wird das umschließende Gebiet berechnet und dieses als Anfragegebiet in Schritt 3 verwendet.
   - Handelt es sich um eine Anfrage nach den nächstgelegenen n Objekten zu einer gegebenen Position? Dann wird die Föderation ein Gebiet um die Position ziehen und dieses als Anfragegebiet in Schritt 3 verwenden. Sollte in Schritt 8 festgestellt werden, dass

keine *n* Objekte gefunden wurden, so wird das Anfragegebiet vergrößert und die Schritte 5 bis 8 wiederholt. Ein effizienter Algorithmus für die Bearbeitung föderierter Nachbarschaftsanfragen findet sich in [SIG+04]. Gibt es zudem noch eine Beschränkung auf bestimmte Objekttypen in der Anfrage? Dann wird daraus eine Typliste gebildet und in Schritt 3 verwendet.
   - Handelt es sich um eine Anfrage, bei der zu gegebenen NOLs weitere Attribute angefragt werden? Dann werden Schritt 3 und 4 übergangen und aus den NOLs eine Serverliste für Schritt 5 gebildet.
   - Wird die Anfrage durch Angabe eines oder mehrerer lokaler Umgebungsmodelle beschränkt? Dann werden Schritt 3 und 4 übergangen und aus den Angaben zu den lokalen Umgebungsmodellen eine Serverliste für Schritt 5 gebildet.
   - Enthält die Anfrage keine solche Beschränkung? Dann wird ein Fehler an die Anwendung zurückgegeben. Die Analyse von Anwendungen hat gezeigt, dass es bei allen benötigten Informationsanfragen möglich ist, die Anfrage mit einer der oben genannten Möglichkeiten einzuschränken. Deswegen ist es nicht notwendig, dass die Föderation uneingeschränkte Anfragen unterstützt (deren Beantwortung die Abfrage aller vorhanden Umgebungsmodellserver erfordern würde!).
3. Das in Schritt 2 ermittelte Anfragegebiet und die evtl. ermittelte Typliste werden nun in eine Anfrage an den räumlichen Verzeichnisdienst umgewandelt und verschickt.
4. Der räumliche Verzeichnisdienst ermittelt, welche Umgebungsmodellserver ein Dienstgebiet haben, das sich mit dem Anfragegebiet überschneidet. Wenn eine Typliste übermittelt wurde, so werden aus der Menge dieser Server diejenigen aussortiert, die keine der in der Liste enthaltenen Typen speichern. Als Ergebnis wird eine Serverliste an die Föderation zurück geschickt.
5. Die Föderation schickt nun die Anfrage der Anwendung an alle Server, die in der Serverliste (aus Schritt 2 oder 4) enthalten sind.
6. Die Umgebungsmodellserver erhalten die Anfrage und bearbeiten diese (für eine detaillierte Semantik der Anfragebearbeitung siehe Kapitel 6.9). Danach schicken sie eine Ergebnismenge von Objekten zurück.

7. Die Föderation wartet, bis sie entweder alle Ergebnismengen von den angefragten Umgebungsmodellservern erhalten hat oder bis eine Zeitüberschreitung auftritt. Sie speichert so lange die erhaltenen Ergebnismengen zwischen.
8. Die Föderation erzeugt nun aus den erhaltenen Ergebnismengen eine Ergebnismenge für die Anwendung. Dabei kann sie mehrere Operationen durchführen:
   - Die Föderation versucht, mehrfach repräsentierte Objekte zu erkennen. Für die Erkennung von mehrfach repräsentierten Objekten ist semantisches Wissen notwendig, das über spezielle Aufbereitungsoperatoren in die Föderation integriert werden kann. Der räumliche Bezug der Objekte liefert gute Hinweise darauf, ob zwei Objekte das selbe Realwelt-Objekt modellieren können oder nicht. Damit wird der übliche n:m-Aufwand der Duplikatserkennung drastisch reduziert.
   - Werden mehrfachrepräsentierte Objekte erkannt, so wird versucht, diese zu verschmelzen. Treten zwei gleiche Attributswerte auf, so wird nur einer beibehalten. Sind die Attributswerte unterschiedlich und existiert ein Algorithmus für die Verschmelzung, so wird dieser angewendet (z.B. „neuester Wert“, „Durchschnitt“,...). Existiert kein solcher Algorithmus, so werden beide Attributswerte zurückgegeben. Ein Ergebnisobjekt kann nun mehrere Objekttypen aufweisen, da der Objekttyp wie ein Attribut behandelt wird.
   - Die Ergebnismenge wird nun auf das Datenschema der Anwendung hin angepasst. Hat die Anwendung in ihrer Anfrage kein Datenschema angegeben, so wird davon ausgegangen, dass sie nur das Standard-Schema unterstützt. Gibt sie erweiterte Schemata an, so wird die Kenntnis dieser Schemata und des Standard-Schemas angenommen. Enthalten die Ergebnismengen Objekte erweiterter Schemata, die der Anwendung nicht bekannt sind, so wird jedes in das speziellste Objekt des Standard-Schemas umgewandelt, von dem es erbt. Zusätzliche Attribute aus dem erweiterten Schema werden weggelassen.
9. Die integrierte Ergebnismenge wird nun an die Anwendung zurückgegeben.
10. Die Anwendung erhält das Ergebnis. Üblicherweise wird sie es in ein internes Format umwandeln und damit weiterarbeiten.

5.3.2 Einfügen von Objekten in das Umgebungsmodell

Das Einfügen von Objekten in das Umgebungsmodell läuft in ähnlichen Schritten ab (siehe Abbildung 6). Die Anwendung schickt eine Menge von Objekten an die Föderation und erhält einen Ergebnisbericht, der pro Objekt entweder eine Erfolgs- oder eine Fehlermeldung enthält.

1. Die Anwendung schickt eine Menge von Objekten an die Föderation.
2. Die Föderation analysiert diese Objekte:
   - Enthalten diese bereits NOLs, so werden daraus die Server-Adressen extrahiert. Für jeden Server wird eine Objektliste erzeugt. Die Schritte 3 und 4 werden übergangen.
   - Ansonsten wird pro Objekt aus seinen räumlichen Attribute ein Anfragegebiet bestimmt, das die Werte aller räumlichen Attribute umschließt. Zudem wird aus den Objekttypen des Objekts eine Typliste gebildet.
   - Enthält ein Objekt keinen Hinweis auf den Server, der es speichern soll *und* kein räumliches Attribut, so wird eine Fehlermeldung erzeugt und in den Ergebnisbericht für die Anwendung eingetragen.
3. Die Anfragegebiete und die Typen werden jeweils nach Objekten getrennt an den räumlichen Verzeichnisdienst übermittelt.
4. Der Verzeichnisdienst ermittelt, welche Server in den jeweiligen Anfragegebieten bereit sind, Objekte des jeweiligen Typs zu speichern und bildet daraus Serverlisten. Um sich zu qualifizieren, muss das Dienstgebiet eines Servers das Anfragegebiet eines Objekts vollständig überdecken. Die Serverlisten werden pro Objekt an die Föderation zurück geliefert.
5. Nun versucht die Föderation, die Objekte bei Servern einzufügen. Objekte, die gleiche Server in ihrer Liste stehen haben, werden gruppiert und gemeinsam an einen Server geschickt. Gibt es zu einem Objekt keinen Server, der es speichern würde, so wird für dieses Objekt eine Fehlermeldung erzeugt und in den Ergebnisbericht eingetragen.
6. Der Server erhält Objekte und versucht diese einzufügen. Tritt dabei ein Fehler auf, so wird pro Objekt eine Fehlermeldung in einen lokalen Ergebnisbericht für die Föderation eingetragen. Ansonsten wird eine Erfolgsmeldung eingetragen. Nachdem alle Objekte abgearbei-

tet wurden, wird der lokale Ergebnisbericht zurück an die Föderation geschickt.

7. Die Föderation wartet auf Ergebnisberichte. Tritt dabei eine Zeitüberschreitung auf, oder enthält Ergebnisbericht eines Servers für ein Objekt eine Fehlermeldung, so versucht sie, Objekte an andere Server ihrer Serverliste zu schicken (zurück zu Schritt 5).
8. Wenn es endgültig nicht gelungen ist, ein Objekt bei einem Server einzufügen, so wird für dieses Objekt ein Fehler in den Bericht eingetragen. Erhält die Föderation Ergebnisdokumente, so trägt sie für die erfolgreich eingefügten Objekte Erfolgsmeldungen in den Bericht ein.
9. Wenn für alle Objekte entweder ein Fehler oder eine Erfolgsmeldung im Bericht steht, wird dieser an die Anwendung zurückgeschickt.

5.3.3 Löschen von Objekten aus dem Umgebungsmodell

Möchte eine Anwendung Objekte aus dem Umgebungsmodell löschen, so spezifiziert sie diese in einer Anfrage, die sie an die Föderation schickt.

Die Löschanfrage wird analog zu Informationsanfragen bearbeitet, mit dem Unterschied, dass die Server keine Ergebnismenge zurück liefern, sondern einen Ergebnisbericht (ähnlich wie beim Einfügen von Objekten), der für jedes betroffene Objekt entweder eine Erfolgsmeldung (das Objekt wurde gelöscht) oder eine Fehlermeldung enthält.

5.3.4 Ändern von Objekten im Umgebungsmodell

Möchte eine Anwendung Objekte im Umgebungsmodell ändern, so spezifiziert sie zunächst diese Objekte über eine Anfrage. Zusätzlich gibt sie noch die Information mit, welche neuen Werte die betroffenen Objekte erhalten sollen.

Die Änderungsanfrage wird analog zu Informationsanfragen bearbeitet, mit dem Unterschied, dass die Server keine Ergebnismenge zurück liefern, sondern einen Ergebnisbericht (ähnlich wie beim Einfügen von Objekten), der für jedes betroffene Objekt entweder eine Erfolgsmeldung (das Objekt wurde geändert) oder eine Fehlermeldung enthält.

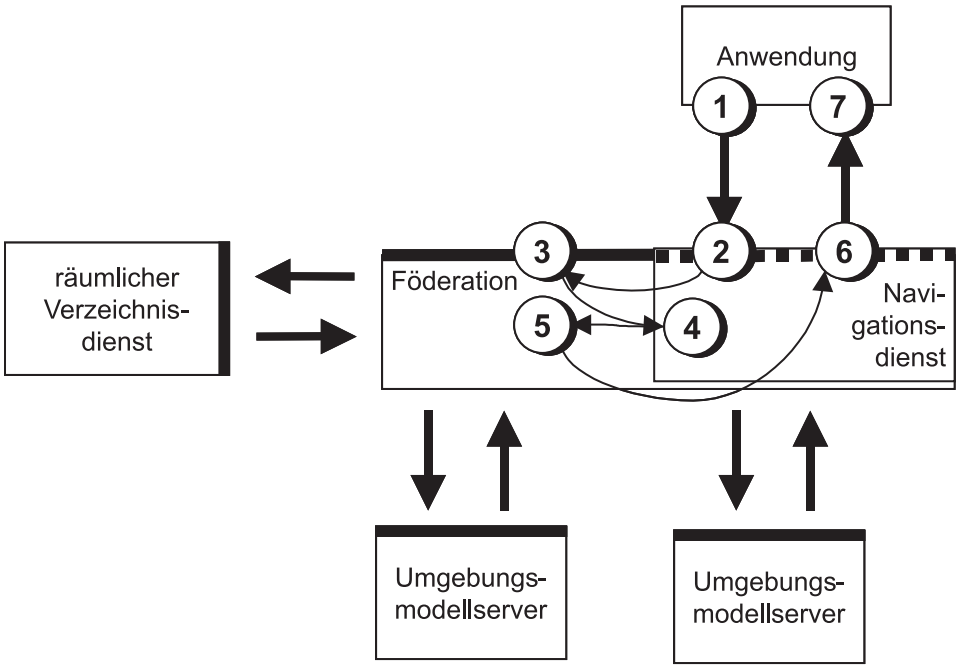

Abbildung 7: Ablauf einer Navigationsanfrage

5.3.5 Anfrage an den Navigationsdienst

Die Schnittstelle des Navigationsdiensts akzeptiert spezielle Anfragen, mit denen eine Anwendung sich eine Route von einem Start- zu einem Zielpunkt über evtl. Zwischenpunkte berechnen lassen kann. Dabei können verschiedene Einschränkungen angegeben werden, z.B. mögliche Verkehrsmittel („Bus“, „LKW 10t“) oder Wege („keine Treppen“). Zudem kann die Anwendung das Ergebnis in unterschiedlichen Formaten erhalten (z.B. als textuelle Beschreibung, als Wegstrecke, als Fahrtanweisungen,...).

Diese Abfrage läuft folgendermaßen ab (siehe Abbildung 7):

1. Die Anwendung schickt eine Navigationsanfrage an den Navigationsdienst.
2. Der Navigationsdienst analysiert die Anfrage und bildet ein Anfragegebiet, in dem die kürzeste Route höchstwahrscheinlich liegen wird. Er schickt eine Informationsanfrage an die Föderation. Da er ein Mehrwertdienst ist, kann er dafür eine interne Schnittstelle verwenden, welche die gleiche Semantik wie die externe Schnittstelle aufweist, jedoch effizienter ist. Er fragt nach Navigationsverbindun-

gen, die mit den Einschränkungen der Navigationsanfrage benutzbar sind.
3. Die Föderation beantwortet diese Anfrage analog zu dem Verfahren in Kapitel 5.3.1.
4. Der Navigationsdienst baut aufgrund des Ergebnisses einen Navigationsgraphen auf, auf dem er die kürzeste Route von Start- zu Zielpunkt berechnet. Je nach gewünschten Rückgabeformat formuliert er gegebenenfalls eine weitere Anfrage an die Föderation, in der er z.B. nach wichtigen Wegpunkten fragt.
5. Auch diese Anfrage wird analog zu Kapitel 5.3.1 beantwortet.
6. Der Navigationsdienst berechnet nun aus der kürzesten Route zusammen mit den evtl. Zusatzinformationen aus Schritt 5 das gewünschte Ergebnisformat und schickt es an die Anwendung.
7. Die Anwendung erhält das Ergebnis, wandelt es gegebenenfalls in eine interne Darstellung um und gibt es an den Benutzer weiter.

5.3.6 Anfrage an den Kartendienst

Die Schnittstelle des Kartendiensts akzeptiert Informationsanfragen zusammen mit einer Spezifikation des gewünschten Ergebnisbildes. Er gibt diese Anfrage über eine interne Schnittstelle an die Föderation weiter, die sie analog zu Kapitel 5.3.1 beantwortet. Die Ergebnismenge wird nun graphisch aufbereitet: über die Spezifikation des Ergebnisbildes bestimmt der Kartendienst, welche Objekte wie eingezeichnet werden sollen und welche Eigenschaften (Format, Auflösung,...) das Ergebnisbild haben soll. Dieses wird berechnet und an die Anwendung zurückgegeben. Die Anwendung kann nun dem Benutzer das Ergebnisbild anzeigen.

5.3.7 Optimierte Abläufe

Die oben beschriebenen Abläufe erfordern viele Kommunikationsverbindungen über Rechnergrenzen hinweg. Dies führt im Allgemeinen zu Einbußen bei der Performanz, was der Anforderung 9 (Effizienz) widerspricht.

Deswegen können Anwendungen, die wissen, von welchen Servern sie Daten erhalten wollen, diese in ihren Anfragen angeben. Damit entfällt für die Föderation die Anfrage an den räumlichen Verzeichnisdienst.

Wenn eine Anwendung für eine bestimmte Anfrage nur die Daten eines bestimmten Servers benötigt, so kann sie diese auch direkt an diesen Server richten. Da Umgebungsmodellserver und Föderation die gleiche Schnittstelle haben, ist dies einfach möglich.

Die Anwendung kann natürlich auch mehrere Server auf diese Weise anfragen, muss die verschiedenen Ergebnisse dann jedoch selbst integrieren.

Im Allgemeinen ist davon auszugehen, dass insbesondere die Föderation Caching-Verfahren einsetzt, um sich Anfragen an den räumlichen Verzeichnisdienst oder die Umgebungsmodellserver zu sparen. Die Mehrwertdienste können dann ebenfalls über eine interne Schnittstelle auf diese Caches zugreifen.

Das Cachen von Umgebungsmodelldaten ist nicht trivial. Da die Datenobjekte häufig nicht komplett abgefragt werden, sind herkömmliche Caching-Verfahren nicht sinnvoll einsetzbar: diese gehen davon aus, dass ein Datenobjekt sich entweder ganz oder gar nicht im Cache befindet. Stattdessen müsste hier prädikatives Caching eingesetzt werden, bei dem zusätzlich zum Cacheinhalt auch die Anfragen gespeichert werden, die dadurch beantwortet wurden, um bei weiteren Anfragen feststellen zu können, welche Teile aus dem Cache beantwortet werden können und welche nicht. Hinzu kommt, dass einige Umgebungsmodelldaten hochdynamisch sind und sich häufig ändern (z.B. die Position mobiler Benutzer), was das Caching ebenfalls erschwert.

Insgesamt ist zu sagen, dass im Bereich Caching von Umgebungsmodelldaten noch großer Forschungsbedarf besteht, den zu decken jedoch den Umfang dieser Arbeit sprengen würde.

Das, was wirklich ist, kann man mit Wörtern ohnehin immer nur annähernd beschreiben; ein Wort ist eben nie genau die Sache selber. (Hans Bemmann)

Kapitel 6: Das Nexus Umgebungsmodell

In diesem Kapitel wird das Nexus Umgebungsmodell vorgestellt. Nach einer kurzen Skizzierung der Entwicklungsgeschichte werden die zentralen Eigenschaften beschrieben, gefolgt von einer Definition der Beschreibungssprachen, die zur Notation des Modells verwendet werden. Für die Darstellung und Verarbeitung des Modells werden zunächst formale Grundlagen gelegt, danach folgt ein Überblick über die Bestandteile des Modells selbst, seine Attributtypen und Objekttypen. Anschließend wird auf die Anfrage und Serialisierung durch die XML-basierte Sprachen AWML und AWQL eingegangen und die technische Repräsentation der Schemainformation durch XML Schemata. Das Kapitel schließt mit den Erweiterungskonzepten des Modells und einem Ausblick auf geplante zukünftige Weiterentwicklungen.

6.1 Entwicklungsgeschichte

Das Umgebungsmodell ist das Ergebnis einer mehrjährigen Projektentwicklung, die noch nicht vollständig abgeschlossen ist. Die Idee wird erstmals in dem visionären Beitrag [HKL+99] unter dem Namen *Augmented World Model* erwähnt, als Integrationsstrategie für verschiedene räumliche Datenmodelle (*Augmented Areas*). Damit sollen verschiedene Anwendungen auf eine einheitliche Sicht über eine einheitliche Schnittstelle zugreifen können. Technisch gesehen wurden die grundlegenden Anforderungen in [Nic+00] formuliert, in einem Bericht, der die erste Architektur und die Schnittstellen der Nexus-Plattform dokumentierte. In der Arbeit [Mes01] wurden diese Anforderungen durch ein typisches Szenario (jemand besucht eine ihm unbekannte Stadt und fragt dort

verschiedene ortsbezogene Informationen ab) in eine erste Version des Modells umgesetzt. Dabei wurde entschieden, aus Gründen der leichteren Verwaltung auf Mehrfachvererbung zu verzichten. Später stellte sich heraus, dass diese Einschränkung jedoch eine sinnvolle Modellierung verschiedener Konzepte erschwerte und die Entscheidung wurde revidiert. In [NM01] wurde erstmals das Modell selbst und eine Methode, wie mit Hilfe des Modells und der Nexus-Plattform Anwendungen entwickelt werden können, veröffentlicht. Eine erweiterte und überarbeitete Version dieser Arbeit findet sich in [NM04].

Im Laufe der Jahre gab es zahlreiche Erweiterungen, mit denen das Umgebungsmodell bezüglich neuer Anwendungsmöglichkeiten ausgebaut wurde. So kam mit [VGH+02] eine detaillierte Modellierung von Straßenverkehrsdaten hinzu, die auf einer Trennung von geographischen und topologischen Daten beruht (siehe Kapitel 6.7.6). In [Grz02] wurden erstmals technische Objekte (hier: Infrarot-Sender) in das Modell integriert, im Jahr 2003 kam eine Erweiterung um Netzinfrastrukturelemente hinzu (siehe Kapitel 6.7.5). Auch für den Bereich ortsbasierter Spiele wurde das Modell erweitert [Fre03] [NHM+04], wie später in Kapitel 8 erläutert wird.

Aktuell in Arbeit befindliche Weiterentwicklungen betreffen Metadaten [HKN+05], Sensoren und Hardware- und Software-Dienste. Auf diese wird in Kapitel 7.3 ein Ausblick gegeben.

6.2 Zentrale Eigenschaften

In diesem Kapitel werden die zentralen Eigenschaften des Umgebungsmodells beschrieben und diskutiert.

6.2.1 Objekt-Basierung

Das Umgebungsmodell ist objekt-basiert: es besteht aus voneinander unterscheidbaren Entitäten (Objekten), die Eigenschaften (Attribute) besitzen können. Objekte sind Instanzen eines Objekttyps, der beschreibt, welche Attribute für Objekte einer bestimmten Art zulässig sind.

Objekte sind für sich gesehen autonom: sie beschreiben einen kohärenten Weltausschnitt (z.B. ein Gebäude oder eine Person) und sind für sich genommen zunächst unabhängig von anderen Objekten, auch wenn sie über Relationen (s.u.) mit solchen verbunden sein können.

Ein Objekt ist im Folgenden immer ein Datenobjekt. Ist eine konkrete Entität in der realen Welt (z.B. ein Gebäude) gemeint, so wird von einem *Realweltobjekt* gesprochen.

Diese Art der Modellierung kommt der menschlichen Wahrnehmung recht nahe. Sie hat mehrere Vorteile:

Durch die Unabhängigkeit der Objekte voneinander können auch kleine Weltausschnitte modelliert und sinnvoll verarbeitet werden. Es können beliebige Submengen aus dem Umgebungsmodell gebildet werden – ob diese zueinander jedoch noch konsistent und vor allem: kohärent sind, muss jedoch gesondert geprüft werden.

Es können heterogene Ergebnismengen gebildet werden. Ein konkreter Weltausschnitt besteht meist aus Objekten verschiedenen Typs: z.B. die für eine Anwendung relevanten Objekte in einem Zimmer wären Modellierungen des Zimmers an sich, der Möbel, der Personen und der installierten Sensoren. Diese Objekte möchte eine Anwendung in einer gemeinsamen Ergebnismenge erhalten.

Allerdings erschwert die Objekt-Basierung die Modellierung größerer Weltausschnitte, wenn die relevanten Realweltobjekte eng miteinander verknüpft sind oder sogar gemeinsame Anteile haben. Wenn z.B. die Zimmer eines Gebäude modelliert werden sollen, ergibt sich das Problem, dass benachbarte Zimmer eine gemeinsame Wand haben. Wenn jedes Zimmer mit seinen Wänden modelliert wird, so werden die Wände selbst mehrfach im Modell vorhanden sein, was zu größerem Datenvolumen und zu Konsistenzproblem führen kann. Wenn Zimmer-Objekte jedoch bereits modellierte Wände anderer Zimmer-Objekte referenzieren, so verliert man den Vorteil der Unabhängigkeit des Objektmodells: einzelne Zimmer-Objekte können nun nicht mehr aus dem Modell gelöst werden, ohne die Referenzen zu verlieren.

Eine Lösungsmöglichkeit für dieses Problem besteht darin, auf Umgebungsmodellserver-Ebene die gemeinsamen Objektteile heraus zu faktorisieren und für mehrere Objekte nur einmal zu speichern. Für die Rückgabe über die Schnittstelle der Plattform muss der Umgebungsmodellserver die Objektteile dann wieder zusammen fügen, so dass sie nur im Ergebnisdokument mehrfach auftreten. Für die Konsistenzwahrung bei Einfüge- und Änderungsoptionen wäre der Umgebungsmodellserver dann selbst verantwortlich.

6.2.2 Identität und Auffindbarkeit von Objekten

Alle Objekte im Umgebungsmodell enthalten einen eindeutigen Bezeichner, genannt *Nexus Object Locator (NOL)*. Durch seinen Aufbau können Objekte mit dem NOL nicht nur eindeutig identifiziert werden, sondern auch ihrem Speicherort zugeordnet werden.

Konzeptionell besteht der NOL aus zwei Teilen: dem Speicherort und einer Objekt-ID. Erst zusammen identifizieren sie ein Objekt weltweit eindeutig.

Im Allgemeinen ist schon die Objekt-ID weltweit eindeutig. Um dies zu erreichen, verwendet jeder Umgebungsmodellserver, der ein neues Objekt anlegt, einen Algorithmus, der aus der weltweit eindeutigen ID seiner Netzwerkkarte (MAC-Adresse) und einem Zeitstempel eine Objekt-ID erzeugt.

Auf einem Umgebungsmodellserver gibt es jede Objekt-ID nur einmal. Anstatt eine neue ID zu erzeugen, kann ein Server jedoch auch bei anderen Servern nachfragen, ob sie bereits eine Repräsentation des selben Realwelt-Objekts verwalten (z.B. über eine räumliche Anfrage an die Föderation). So können kooperierende Umgebungsmodellserver die Objekt-ID dazu nutzen, Mehrfachrepräsentationen zu kennzeichnen.

Wenn die Föderation in einer Ergebnismenge Objekte mit gleicher Objekt-ID findet, kann sie davon ausgehen, dass es sich um eine Mehrfachrepräsentation handelt und die beiden Objekte verschmelzen.

Anwendungen können den NOL auch dazu nutzen, um zu einem Objekt weitere Informationen zu erhalten (siehe Schritt 2 von Kapitel 5.3.1).

Zudem ist der NOL ist eine Voraussetzung für Relationen (siehe Kapitel 6.2.9 und 7.3.2), da über ihn die teilnehmenden Objekte an einer Relation identifiziert werden können.

Die Spezifikation des NOL findet sich in Kapitel 6.6.1.

6.2.3 Objekttypen

Die Objekte des Umgebungsmodells werden nicht frei modelliert, sondern entsprechen in ihrer Struktur einem oder mehreren Objekttypen (i.a. auch Klasse genannt). Die Objekte sind also Instanzen ihres Objekttyps. Die Gesamtheit aller Objekttypen für ein Umgebungsmodell wird als Schema bezeichnet.

Der Objekttyp legt Struktur und Aufbau seiner Instanzen fest. Im Unterschied zu objektorientierter Programmierung haben die Objekte des Umgebungsmodells keine Methoden, d.h. kapseln keinen ausführbaren Code. Sie dienen der statischen Repräsentation von Realweltobjekten oder digitalen Objekten und modellieren deren Eigenschaften, nicht deren Verhalten.

Es lässt sich nicht immer eine eindeutige Beziehung zwischen Realweltobjekt und Objekttyp herstellen. Verschiedene Menschen können beim Modellieren zu unterschiedlichen Ergebnissen kommen, insbesondere, wenn sie ihr Modell für verschiedene Zwecke erstellen. So reicht es für eine grobe Landkarte aus, ein Gebäude nur mit seinem Umriss zu erfassen, während für touristische Anwendung das selbe Gebäude als Kirche (mit Attributen für geschichtliche Informationen etc.) modelliert wird. Bei der Zusammenführung der Modelle verschiedener Datenanbieter kann es dabei zu Inkonsistenzen kommen. Um diese aufzulösen, können Objekte im Umgebungsmodell mehrere Objekttypen haben.

6.2.4 Vererbung zwischen Objekttypen

Zwischen den Objekttypen besteht eine Vererbungsbeziehung, eine “ist-ein-Relation”. Erbt ein Objekttyp A von einem Objekttyp B, so besitzt A sämtliche Eigenschaften von B (“A ist ein B”), kann aber auch noch eigene Eigenschaften hinzufügen. Diese Art der Beziehung wird auch als Spezialisierung bezeichnet.

Die Vererbungsbeziehung der Objekttypen im Umgebungsmodell kann auf verschiedene Weise ausgenutzt werden:

- Objekttyp-Selektion bei bekanntem Schema: Eine Anwendung kann gleichartige Objekte aufgrund ihrer Objekttypen auswählen. Wenn sie alle Objekte vom Typ “Raum” anfragt, erhält sie auch alle Objekte, die Instanzen von einem Typ sind, der vom Typ “Raum” erbt, also Besprechungsräume, Seminarräume, Büroräume oder Flure. Damit die

Anwendung mit dem Ergebnis auch umgehen kann, muss sie das Schema kennen.
- Objektumwandlung bei teilweise unbekanntem Schema: Wenn die Anwendung ein allgemeines Schema kennt, das Schema eines spezialisierten (d.h. vom allgemeinem Schema ererbten) Modells jedoch nicht, kann sie trotzdem Daten erhalten, die sie verarbeiten kann. Die Objekte des spezialisierten Modells werden entlang der Objekttypenhierarchie nach oben umgewandelt (*"upcast"*). Zusätzliche Eigenschaften der spezialisierten Objekte werden weggelassen. Die Anwendung erhält Objekte des allgemeinen Schemas, obwohl sie ursprünglich im spezialisierten Schema vorliegen.

Bei der Schemamodellierung tritt jedoch das Problem auf, dass ein Objekttyp von mehreren Objekttypen erben sollte. So ist z.B. eine Kirche sicherlich ein Gebäude, eine historische Kirche jedoch zugleich eine spezielle Kirche als auch eine Sehenswürdigkeit (hat Öffnungszeiten).

Das Schema des Umgebungsmodells unterstützt hierfür Mehrfachvererbung.

6.2.5 Eindeutigkeit der Attribute

Ortsbezogene Anwendungen benötigen häufig heterogene Modellausschnitte, da z.B. in digitale Karten Objekte verschiedener Typen eingezeichnet werden sollen. Es wäre nun sehr aufwändig, diese benötigen Daten typweise einzeln aus dem Umgebungsmodell abzufragen. Um Anfragen über Objekttypgrenzen hinweg zu ermöglichen, wird eine Eindeutigkeit der Attribute festgelegt. Das bedeutet, dass es eine feste Menge von Attributen gibt, die in einem Attributsschema (Kapitel 6.6.2) Datentypen zugeordnet werden. Diese Attribute können nun von den Objekttypen verwendet werden, um das Aussehen von Objekten zu spezifizieren. Sie können jedoch nicht mehr in ihrem Datentyp und ihrer grundlegenden Semantik geändert werden: besitzt ein Objekt ein Attribut **name**, so bezeichnet dieses den Namen des Objekts (und nicht etwas anderes). Objekttypen spannen also keinen eigenen Namensraum auf.

Dies hat den Vorteil, dass Anwendungen bei einer Anfrage nicht spezifizieren müssen, auf welche Objekttypen sich ihre Auswahl bezieht. Sie können Operatoren über Attribute definieren, die auch für alle Objekte, die diese Attribute enthalten, ausgewertet werden können. Eine formale Definition dieser Eigenschaft findet sich in Kapitel 6.4.

6.2.6 Ortsbezug des Umgebungsmodells

Das Umgebungsmodell ist bereits vom Entwurf her ortsbezogen, um Anforderung 1 (Ortsbasierter Zugriff auf Information) und Anforderung 9 (Effizienz) erfüllen zu können. Es ist also daraufhin optimiert ist, über den Ort als primären Zugriffspfad abgefragt zu werden.

Dazu werden alle Objekte, bei denen es semantisch sinnvoll ist, mit einer Ortsinformation versehen. Bei Objekten der physischen Welt ist dies einfach durch ihren Standort gegeben. Für digitale Objekte (wie z.B. Webseiten) kann häufig ein Ortsbezug über ihre Relevanz hergestellt werden: der Ort einer Webseite wird informal definiert durch das Gebiet, in dem sie von Interesse ist (nach der Meinung des Modellierers). Als Ort einer ortsbezogenen Anwendung wird das Gebiet angenommen, für das sie konzipiert wurde.

Objekte ohne direkten Ortsbezug (z.B. eine Anwendung, die weltweit eingesetzt werden kann, allgemeine Dokumente,...) können auch im Umgebungsmodell modelliert werden. Allerdings ist für diese Objekte kein Zugriff über den Ort möglich. Deswegen muss eine Anwendung für diese Objekte den NOL kennen, um über die Nexus-Plattform auf die Objekte zugreifen zu können – entweder vorab, oder der NOL wurde aus einer früheren Anfrage über Relationen in Objekten mit Ortsbezug ermittelt (siehe Kapitel 6.2.9).

6.2.7 Das Lokationsmodell

Das Lokationsmodell dient als Grundlage für den Ortsbezug. Dadurch wird festgelegt, wie der Raum selbst modelliert wird. Grundsätzlich könnte dies entweder implizit oder explizit geschehen.

Implizit bedeutet, dass der Raum nur durch räumliche Beziehungen (*vor, hinter, neben,...*) zwischen Objekten entsteht und nicht selbst modelliert wird. Man spricht auch von topologischen Lokationsmodellen oder räumlichen Kalkülen [RC89].

Für das Umgebungsmodell wurde jedoch ein geometrisches Lokationsmodell gewählt. Hierbei wird der Raum selbst als Fläche modelliert, auf der die Objekte angeordnet werden. Die räumlichen Beziehungen ergeben sich daraus implizit. Mit diesem Ansatz ist eine größere Unabhängig-

keit zwischen Objekten und verschiedenen Teilen des Modells möglich, da sie das geometrische Lokationsmodell als gemeinsame Referenz verwenden können.

Die Geometrien im Umgebungsmodell werden nach der Simple Feature Specification der OGC angegeben [SFS]. Dadurch können Punkte, Linien, Polygone und zusammengesetzte Geometrien dargestellt werden, jedoch keine Kreise, die durch Polygone angenähert werden müssen. Die Nachteile dieser Einschränkung (komplexere Modellierung) wogen jedoch den Vorteil nicht auf, auf einen etablierten Standard mit existierenden Implementierungen zurückgreifen zu können.

Um die Geometrien im tatsächlichen Raum zu verankern, benötigen sie ein zugehöriges Koordinatensystem. Hierbei gibt es grundsätzlich den Unterschied zwischen kartesischen (z.B. Gauss-Krüger) und sphärischen Koordinatensystemen (z.B. [WGS84]). Kartesische Koordinatensysteme sind flächen- und winkeltreu und ermöglichen einfache geometrische Berechnungen. Da die Erde jedoch rund ist [Ari93], stellen sie eine Verzerrung der tatsächlichen Verhältnisse dar. Zu einem kartesischen Koordinatensystem gehört daher immer ein Referenzgebiet, in dem die Verzerrung unter einem gegebenen Schwellenwert liegt. Außerhalb dieses Gebiets sind die Werte des Koordinatensystems nicht gültig.

Sphärische Koordinatensysteme dagegen bezeichnen Punkte auf der Weltkugel mit Längen- und Breitengraden. Dadurch können sie weltweit gültig sein. Ihr Nachteil ist jedoch, dass sie die geometrischen Figuren auf die Weltkugel projizieren und Berechnungen damit aufwändig werden.

Deswegen ist das Umgebungsmodell unabhängig von einem konkreten Koordinatensystem. In der Datenbank der *European Petroleum Survey Group* [EPSG] sind über 2800 verschiedene Koordinatensysteme gespeichert und Transformationen zwischen ihnen angegeben. Über einen EPSG-Code wird im Umgebungsmodell gekennzeichnet, welches Koordinatensystem verwendet wird. Mit Hilfe von Softwarebibliotheken (z.B. [SHG+04]) können Koordinaten verschiedener Koordinatensysteme für die Anwendung transparent umgewandelt und verrechnet werden.

6.2.8 Lokale Umgebungsmodelle

Lokale Umgebungsmodelle bilden die Grundlage für das Umgebungsmodell. Man kann wohl kaum davon ausgehen – und es wäre auch nicht wünschenswert – dass eine zentrale Instanz (oder Organisation) ein komplexes Modell der Welt aufstellt, verwaltet und konsistent hält. Deswegen ermöglicht es die Nexus-Plattform, über zahlreiche lokale Umgebungsmodelle zu föderieren, die von verschiedenen Anbietern bereitgestellt werden.

Um diese Föderation und auch eine Zusammenführung der lokalen Umgebungsmodelle zu ermöglichen, müssen diese bestimmten Kriterien genügen:

- Dienstgebiet: Ein lokales Umgebungsmodell liegt innerhalb eines eingeschränkten geographischen Gebiets. Dieses Gebiet kann beliebig gewählt werden, aber es muss alle geographischen Attribute der im Modell enthaltenen Objekte vollständig umschließen. Das Dienstgebiet kann entweder im Voraus festgelegt werden (dann können im Modell nur Objekte gespeichert werden, deren geographische Attribute im Modell liegen), oder es wird aus einer gegebenen Menge von Objekten als Hülle um die geographischen Attribute berechnet. Dienstgebiete verschiedener lokaler Umgebungsmodelle dürfen sich überlappen. Die Position und die Ausmaße des Dienstgebiets werden als Metadaten in der Nexus-Plattform im Area Service Register gespeichert (siehe Kapitel 5.2.2). Das Dienstgebiet kann prinzipiell beliebig groß sein, nur ist es für die Verarbeitung sinnvoll, es so klein wie möglich zu wählen, da somit unnötige Anfragen (das sind Anfragen für Gebiete, in denen keine Objekte gespeichert werden) vermieden werden können.
- Objekttypen: Ein lokales Umgebungsmodell enthält nur Objekte, die Instanzen einer Menge von bestimmten Objekttypen sind. Die Objekttypen können entweder aus dem Standard-Schema oder einem oder mehreren erweiterten Schemata stammen. Es können alle Objekttypen eines Schemas von einem lokalen Umgebungsmodell in seiner Objekttypenmenge enthalten sein. Die Liste der Objekttypen wird als Metadatum in der Nexus-Plattform im Area Service Register gespeichert (siehe Kapitel 5.2.2). Analog zum Dienstgebiet ist es für die Verarbeitung sinnvoll, die Objekttypenmenge möglichst klein zu wählen, da somit unnötige Anfragen vermieden werden können.

- Konsistenz: Ein lokales Umgebungsmodell ist in sich konsistent. Das bedeutet, dass es keine Mehrfachrepräsentationen von Objekten gibt, sprich, jedes Realwelt- oder virtuelles Objekt ist nur einmal im Modell enthalten.
- Speicherung: Ein lokales Umgebungsmodell wird komplett auf einem logischen Speicher verwaltet. In der Nexus-Plattform befindet sich ein lokales Umgebungsmodell also komplett auf einem räumlichen Umgebungsmodellserver, der jedoch mehrere lokale Umgebungsmodelle verwalten kann.
- Kooperative lokale Umgebungsmodelle: Mehrere lokale Umgebungsmodelle können kooperieren. Das bedeutet, dass sie über Benachrichtigungsmechanismen und gegenseitige Anfragen sicherstellen, dass bei Mehrfachrepräsentationen die Objektidentität angeglichen wird. Wird in ein solches Umgebungsmodell eine neue Repräsentation eines Realwelt-Objekts eingefügt, so wird zunächst überprüft, ob andere Umgebungsmodelle bereits eine Repräsentation dieses Realweltobjekts enthalten. In diesem Fall bekommt das neue Objekt die Objektidentität der bereits existierenden Repräsentationen. Solche Objekte haben jedoch immer noch eindeutige NOLs, da der Speicherort (und damit die „Adresse“ des lokalen Umgebungsmodells) unterschiedlich ist.

In Abbildung 8 ist ein Beispiel zu sehen: drei verschiedene räumliche Umgebungsmodellserver verwalten jeweils ein lokales Umgebungsmodell (LUM1 bis LUM3). LUM 1 enthält Virtuelle Litfaßsäulen (VIT) und ausgesuchte Bauwerke in großer Detailgenauigkeit, LUM 2 speichert Straßen und Gebäude (dabei auch Bauwerke von LUM 1) in einer vereinfachten Form, und LUM 3 detailliert eines der Gebäude aus LUM2 und enthält Innenräume. Das Area Service Register (ASR) speichert die relevanten Metadaten über die lokalen Umgebungsmodelle.

Durch die Föderation wird die Verteilung des Umgebungsmodells auf verschiedene lokale Umgebungsmodelle vor der Anwendung verborgen. Über die Struktur der NOLs kann die Anwendung jedoch zuordnen, welches Objekt aus welchem lokalen Umgebungsmodell stammt, und wo dieses verwaltet wird. Damit ist es möglich, dass Anwendungen ihre Anfragen auch auf einzelne lokale Umgebungsmodelle einschränken können.

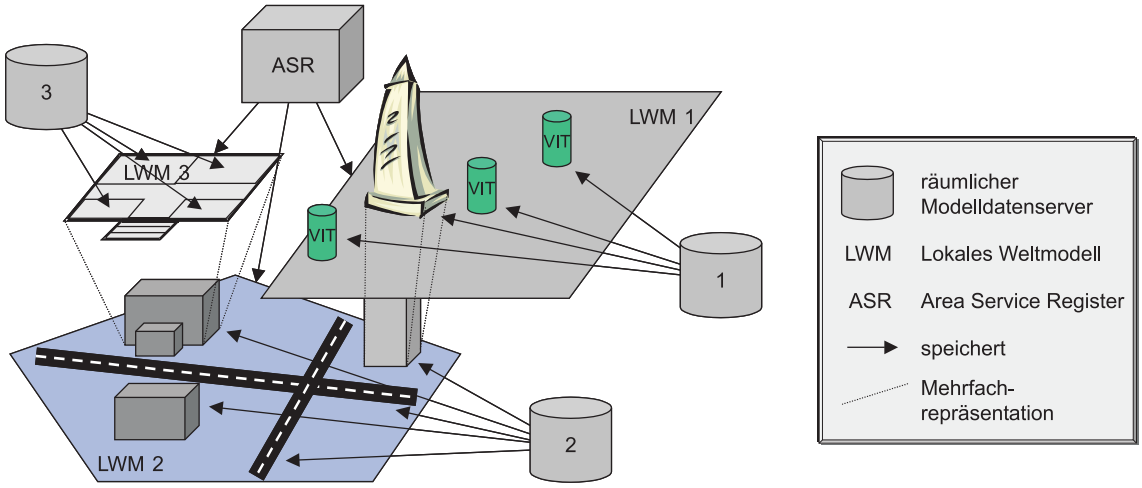

Abbildung 8: Lokale Umgebungsmodelle

6.2.9 Relationen zwischen Objekten

Im Umgebungsmodell sollen auch Beziehungen (Relationen) zwischen Objekten modelliert werden können. Dabei kann unterschieden werden zwischen räumlichen und nicht-räumlichen Relationen.

Räumliche Relationen, z.B. *ist-enthalten-in, liegt-neben* oder *überlappt-sich-mit* bilden die Grundlage von rein topologischen Lokationsmodellen wie z.B. [RC89]. Da dem Umgebungsmodell jedoch ein geometrisches Lokationsmodell zugrunde liegt (siehe Kapitel 6.2.7), ist eine explizite Modellierung räumlicher Relationen meist nicht notwendig, da diese bereits durch die geometrischen Daten gegeben sind und über räumliche Anfragen abgefragt werden können.

Beispiele für nicht-räumliche Relationen sind *gehört, ist-gemietet-von, ist-Partnerstadt-von.* Diese können explizit angegeben werden, da sie nicht aus dem Lokationsmodell abgeleitet werden können.

In seiner derzeitigen Version unterstützt das Umgebungsmodell ausschließlich unidirektionale Relationen zwischen Objekten. Dieses Modell hat sich im WWW bewährt und erfüllt auch die Anforderung 13 (Einfachheit des Umgebungsmodells). Damit kann ein Objekt auf ein anderes verweisen. Bidirektionale Relationen müssen durch zwei unidirektionale Relationen modelliert werden. Der Grund für diese Entscheidung liegt in

der Verwaltung des Modells: da es über mehrere autonome Server verteilt ist und es keine zentrale Stelle geben soll, die über die Konsistenz zwischen Modellen wacht, ist es bei Relationen zwischen Objekten verschiedener Server nicht möglich, zu gewährleisten, dass eine bidirektionale Relation auch auf beiden Servern gespeichert wird. Dies ist aber notwendig, damit die Relationsinformation auch dann gefunden wird, wenn nur das Dienstgebiet eines der beiden Server in einer Anfrage enthalten ist.

Für die Zukunft des Umgebungsmodells wird derzeit ein Relationenmodell diskutiert, bei dem die Relationen eigenständige Objekte sind, die auch auf einem dritten Server gespeichert werden können, ähnlich den *extended links* in [XLink]. Damit diese bei räumlichen Anfragen auch gefunden werden, benötigen sie entweder selbst ein räumliches Attribut, oder es wird zusätzliche Plattformunterstützung benötigt. Je mehr Relationen das Modell enthält, um so geringer wird auch die Unabhängigkeit der einzelnen Objekte. Diese Konzepte und ihre Auswirkungen sind noch nicht evaluiert und deswegen nicht Teil dieser Arbeit. Sie werden überblicksartig in Kapitel 7.3.4 erläutert.

6.2.10 Unterscheidung: statisch, dynamisch und mobil

Damit die Daten des Umgebungsmodells effizient verwaltet werden können, kann es notwendig sein, verschiedene Charakteristika von Attributen zu unterscheiden.

Statische Attribute sind solche, die sich sehr viel seltener ändern, als sie abgefragt werden. Sie werden in der Regel einmal angelegt und ändern sich nicht (oder sehr selten), bis das dazugehörige Objekt wieder gelöscht wird. Beispiele sind die Umrisse eines Hauses oder der Name einer Person.

Dynamische Attribute ändern sich weitaus häufiger, als sie abgefragt werden. Ihre Werte werden meist automatisch erfasst (von Sensoren) oder berechnet. Die meisten Anfragen sind nur an aktuellen Werten dynamischer Attribute interessiert (z.B. der Temperatur in einem Raum). Sofern keine Historie angelegt werden soll, ist daher keine Persistenz der Daten notwendig, da nach einem Systemausfall bald wieder der neueste Wert geliefert werden kann.

Ein Speziallfall der dynamischen Attribute sind *mobile* Attribute. Diese enthalten eine dynamische Ortsinformation. Der häufigste Fall hierfür ist die Position eines mobilen Objekts, z.B. einer Person oder eines Autos. Da der Ort das primäre Auswahlkriterium im Umgebungsmodell darstellt, erfordert die Verwaltung mobiler Attribute eine besondere Behandlung.

Das Umgebungsmodell trägt diesem Unterschied in folgender Weise Rechnung:

Es wird auf Attributsebene grundsätzlich nicht zwischen statischen und dynamischen, aber nicht mobilen Attributen unterschieden. Der Attributswert wird auf gleiche Art und Weise modelliert, unabhängig von seiner Änderungshäufigkeit, denn auf Schemaebene kann nicht eindeutig festgelegt werden, ob ein Attribut dynamisch ist oder nicht. In zukünftigen Modellerweiterungen ist jedoch angedacht, über Metadaten zusätzliche Informationen zur Dynamik konkreter Attributsinstanzen angeben zu können (siehe Kapitel 7.3.1).

Auf Objektebene wird jedoch zwischen mobilen und stationären Objekten unterschieden. Mobile Objekte sind solche, die mindestens ein mobiles Attribut besitzen, nämlich die Position. Sie werden in einer speziellen Objekthierarchie zusammengefasst und sind damit für Anwendungen auch direkt auswählbar (z.B. *alle mobilen Objekte im Umkreis von 10m*).

6.2.11 Unterscheidung: real und virtuell

Das Umgebungsmodell erfüllt die Anforderung 1 (Ortsbasierter Zugriff auf Information) dadurch, dass es ein Modell der physischen Welt mit digitalen Informationen anreichert und beides für die Anwendung zugreifbar macht. Es enthält sowohl Repräsentationen von physischen Dingen wie Gebäuden oder Personen als auch digitale Informationen wie Dokumente oder Anwendungen selbst. Da es objektbasiert ist, werden beide Arten von Daten in Objekten abgebildet. Wir sprechen von realen Objekten (d.h. sie existieren in der physischen Welt) bzw. virtuellen Objekten (d.h. sie existieren dort nicht). Virtuelle Objekte sind Metaphern für digitale Informationen: auch wenn diese in der physischen Welt nicht existieren, so kann man sie sich als Benutzer oder Entwickler ortsbezogener Anwendungen als dort verankert vorstellen. Um virtuelle Objekte wahrnehmen zu können, erfordert es jedoch immer eine Anwendung, die auf die Nexus-Plattform zugreift und virtuelle Objekte in geeigneter Form sichtbar macht. So können *Augmented Reality*-Anwendungen virtu-

elle Objekte in das Sichtfeld des Benutzers einblenden und diese tatsächlich an ihrem Ort sichtbar machen. Das gleiche gilt für die Interaktion: Benutzer können mit realen Objekten direkt interagieren (und Anwendungen können davon nur über Sensorik Kenntnis erlangen), für virtuelle Objekte erfordert es jedoch eine entsprechende Anwendung.

Die Unterscheidung von virtuellen und realen Objekten ist für die Verwaltung des Umgebungsmodells unerheblich. Für die Komponenten der Nexus-Plattform ist es egal, ob das Datenobjekt, das sie speichern und auswerten, ein physisches Objekt darstellt oder nicht. Für Anwendungen ist es jedoch eine wichtige Information, ob das Objekt virtuell oder real ist, sprich, ob ihr Benutzer dagegen laufen kann oder nicht.

Die Unterscheidung kann auch nicht durchgängig an bestimmten Objekttypen festgemacht werden. Man kann sich durchaus virtuelle Gebäude vorstellen (die noch in Planung sind) und auch reale Dokumente (Ausdrucke auf Papier). Deswegen wird diese Unterscheidung im Umgebungsmodell nur durch ein Attribut festgelegt, das den Wert *virtual* oder *real* haben kann und das jedes Objekt festlegen muss.

6.2.12 Standard-Schema und erweiterte Schemata

Die Definition eines globalen Schemas für das Umgebungsmodell widerspricht der Anforderung 12 (Erweiterbarkeit des Umgebungsmodells). Deswegen besteht das Schema des Umgebungsmodells aus einem Standard-Schema und einer Menge erweiterter Schemata.

Das Standard-Schema definiert eine gemeinsame Weltsicht für die Anwedungsdomäne. Welche Objekttypen und Attributtypen dort enthalten sind, ist ein Ergebnis der Szenarien-Analyse.

Wenn Daten integriert werden sollen, die bisher im Standard-Schema noch nicht modelliert sind, so gibt es die Möglichkeit, ein erweitertes Schema zu definieren. Dabei werden im erweiterten Schema neue Objekttypen definiert, die von mindestens einem Objekttyp des Standard-Schemas erben müssen (siehe Abbildung 9). Damit können neue Anwendungen, sofern sie von dem erweiterten Schema Kenntnis haben, auf diese zusätzlichen Daten zugreifen. Doch auch bestehende Anwendungen profitieren von einem erweiterten Schema: da eine Vererbungsbeziehung zu bekannten Objekttypen angegeben ist, kann die Nexus-Plattform bei einer Anfrage auch die Objekte eines erweiterten Schemas

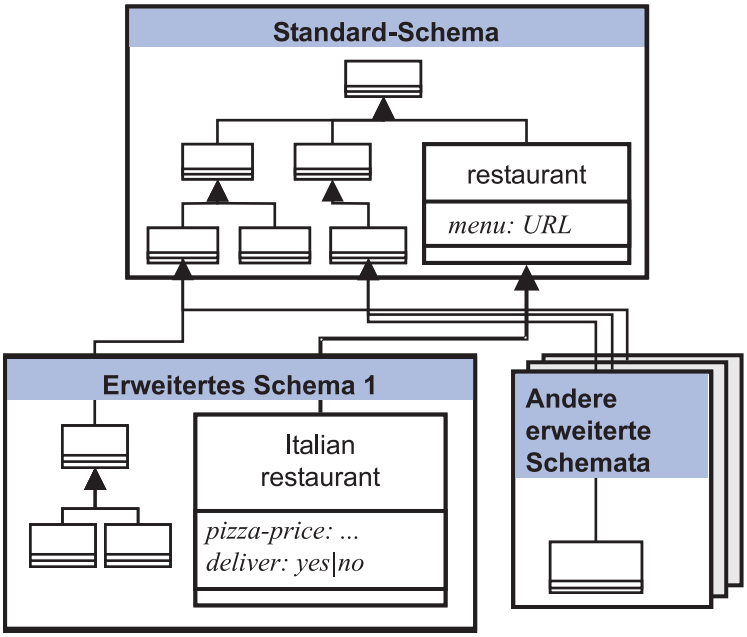

Abbildung 9: Standard-Schema und erweiterte Schemata

zurückgeben. Da die Anwendung jedoch nicht dafür entwickelt wurde, die Daten des erweiterten Schemas zu verarbeiten, werden solche Objekte transformiert, und zwar in eine Instanz des speziellsten Typs des Standard-Schemas. Attribute, die durch das erweiterte Schema definiert wurden, werden dabei weggelassen (vgl. auch Schritt 8 in Kapitel 5.3.1).

Durch diesen Mechanismus kann sich auch das Standard-Schema selbst weiterentwickeln. Für alte Anwendungen erscheinen neue Versionen des Standard-Schemas als erweitertes Schema.

Details zur Definition erweiterter Schemata findet sich in Kapitel 7.1.

6.3 Beschreibungssprachen

In diesem Kapitel werden die Notationen vorgestellt, mit dem das Umgebungsmodell in diesem Bericht beschrieben wird. Wenn man das Umgebungsmodell als Ontologie ansieht, handelt es sich hierbei um die Meta-Ontologie, sprich das Vokabular, mit dem die Ontologie – also das Umgebungsmodell – beschrieben wird (siehe Kapitel 3.2.3). Für die Bezeichnungen des Umgebungsmodells wurde Englisch als Sprache gewählt.

6.3.1 Begriffsdefinitionen

Objekt

Ein Objekt – oder Datenobjekt – ist eine unterscheidbare Entität im Umgebungsmodell. Es wird unterschieden zwischen *realen* und *virtuellen* Objekten. Reale Objekte sind Abbilder von Dingen in der physischen Welt (z.B. Gebäude oder Personen). Virtuelle Objekte sind Abbilder von digitalen Informationen oder Programmen (z.B. Web-Seiten oder Anwendungen). Ein Objekt folgt in seiner Struktur einem oder mehreren Objekttypen und besitzt Attribute.

Objekttyp

Ein Objekttyp beschreibt die Struktur und den Aufbau gleichartiger Objekte. Er legt fest, welche Attribute Objekte dieses Objekttyps haben können und ob diese erforderlich oder optional sind.

Attribut

Ein Attribut ist eine Eigenschaft eines Objekts. Objekte können mehrere Attribute haben. Ein Attribut folgt in seiner Struktur einem Attributtyp. Attribute können *erforderlich* oder *optional* sein. Erforderliche Attribute müssen in einem Objekt vorhanden sein (wenn auch nicht in jeder Ergebnismenge einer Anfrage). Optionale Attribute können in einem Objekt vorhanden sein.

Attributtyp

Ein Attributtyp beschreibt die Struktur und den Aufbau gleichartiger Attribute. Im Umgebungsmodell besteht eine feste Beziehung zwischen einem Attribut und seinem Attributtyp, sprich Attribute gleichen Namens haben stets den gleichen Attributtyp. Es gibt *einfache* und *komplexe* Attributtypen. Einfache Attributtypen beschreiben genau einen Wert, den das Attribut annehmen kann (z.B. *extent*). Komplexe Attributtypen bestehen aus mehreren Werten, deren Name und Struktur vom Attributtyp festgelegt wird (z.B. *address*, besteht aus *street, city, zipcode, state und country*).

Vererbung

Zwischen den Objekttypen des Umgebungsmodells besteht eine Vererbungsbeziehung. Diese ist eine *„is-a“*-Relation: sie bedeutet, dass Objekte eines Kind-Objekttyps auch als Objekte eines Vater-Objekt-

typs angesehen werden können. Das Umgebungsmodell erlaubt *Mehrfachvererbung*: dies bedeutet, dass ein Objekttyp mehrere Väter in der Vererbungshierarchie besitzen darf. Es sind jedoch keine Ringbeziehungen erlaubt, d.h. die Vererbungsrelation ist eine Halbordnung (siehe Kapitel 6.4.1, *(vii)-(ix)*).

6.3.2 Graphische Notation

Für die graphische Notation wird die *Uniform Modeling Language (UML)* [BRJ99] verwendet (siehe Abbildung 10). Dabei werden die Objekttypen als Klassen dargestellt, die Attribute als Klassenattribute. Die Vererbung zwischen Objekttypen wird durch Generalisierungspfeile angegeben. Erforderliche Attribute werden durch Fettdruck von optionalen Attributen unterschieden.

Um in der graphischen Notation Platz zu sparen, werden in den Abbildungen dieses Kapitels Abkürzungen für die Attributtypen verwendet. Dabei wird **Nexus*AttributeType** abgekürzt durch *****, also **NexusAddressAttributeType** durch **Address** oder **NexusStringAttributeType** durch **String**.

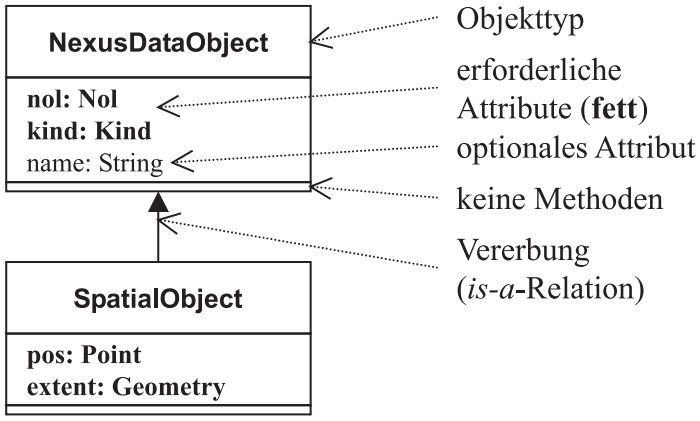

Abbildung 10: Graphische Notation (UML)

6.3.3 Technische Notation

Für die Verarbeitung des Umgebungsmodells wird eine XML-Darstellung verwendet. Das Datenschema (Objekt- und Attributtypen) wird in mehreren XML Schemata [XML Schema] Dateien festgelegt. Objekte können in einer XML Notation serialisiert werden, die *Augmented World*

Modeling Language (AWML). Anfragen an das Umgebungsmodell können in einer räumlichen Anfragesprache formuliert werden, die ebenfalls auf XML beruht, die *Augmented World Query Language* (AWQL). Die Details dieser Sprachen werden in den Kapiteln 6.5, 6.8 und 6.9 beschrieben.

6.3.4 Formale Notation

In Kapitel 6.4 wird eine formale Notation verwendet, um zentrale Eigenschaften des Modells zu definieren und abzuleiten. Dabei werden Elemente der Schema-Ebene mit kleinen griechischen Buchstaben bezeichnet, Mengen dieser Elemente mit großen griechischen Buchstaben und Elemente der Instanz-Ebene (Objekte) im wesentlichen mit kleinen bzw. großen römischen Buchstaben (Tabelle 2):

Tabelle 2: Bezeichner der formalen Notation

Zeichen	Bedeutet	Ebene
α	Attribut	Schema
β	Attributtyp	Schema
χ	Objekttyp	Schema
δ	Objekttypname	Schema
κ	Attributschema	Schema
λ	Objektschema	Schema
A_{erf}	erforderliche Attribute	Schema
A_{opt}	optionale Attribute	Schema
v	Wert eines Attributs	Instanz
o	Objekt	Instanz
A_o	Attribute des Objekts o	Instanz
$\chi' \angle \chi$	Vererbung	Schema
$\chi \ll o$	Instanziierung	Schema/Instanz
σ	Selektion	Instanz
π	Projektion	Instanz
F	Filter	Instanz

6.4 Formale Grundlagen

In diesem Kapitel werden nun einige zentrale Eigenschaften der Modellierung formal eingeführt. Dabei werden die Bezeichner gemäß Tabelle 2 verwendet. Vorhergehende Definitionen, die in späteren Formeln verwendet werden, werden durch Verweis auf die Definitionsnummer (kleine römische Ziffern) auf der rechten Seite referenziert.

6.4.1 Definition eines Schemas

Ein Schema λ sei ein Tripel aus einem Attributsschema κ, einer Menge X von Objekttypen $\chi \in X$ und einer Relation $\angle$ (Vererbung):

(i) $$\lambda = <\kappa, X, \angle >$$

Es gibt eine Menge A, die Attribute α enthält. Das Attributschema κ ordnet jedem α einen Attributtyp β zu (injektive Funktion):

(ii) $$\kappa\colon A_k \rightarrow B \qquad | \alpha \in A_\kappa, \beta \in B$$

Attributtypen sind Datentypen (z.B. *string, boolean* oder *integer).* Für einen konkreten Wert *v* kann geprüft werden, ob er vom Datentyp β ist (*(xii)*).

Ein Objekttyp χ sei ein Tripel aus einem Objekttypnamen δ, einer Menge erforderlicher Attribute A_{erf} und einer Menge optionaler Attribute A_{opt}:

(iii) $$\chi = < \delta, A_{erf}, A_{opt} > \qquad | A_{erf,} A_{opt} \subseteq A$$

Dabei gilt, dass ein Attribut eines Objekttyps entweder erforderlich oder optional ist (und nicht beides):

(iv) $$A_{erf,\chi} \cap A_{opt,\chi} = \varnothing$$

Die Attribute α der Objekttypen χ müssen Teil der Attributmenge A sein:

(v) $$\forall \chi \in X\colon A_{erf} \in A \wedge A_{opt} \in A \qquad | \textit{(iii)}$$

Die Vererbung $\angle$ ist eine Relation zwischen Objekttypen. Es gilt, dass ein erbender Objekttyp die Attributmengen seines Vorgängers erweitern kann sowie optionale Attribute erforderlich machen. Erforderliche Attribute können beim Vererbungsschritt jedoch niemals optional werden:

(vi) $$\chi' \angle \chi \Rightarrow A_{erf,\chi'} \subseteq A_{erf,\chi} \wedge \forall \alpha \in A_{opt,\chi'} : \alpha \in A_{opt,\chi} \vee \alpha \in A_{erf,\chi}$$

Die Vererbungsrelation hat die Semantik einer Spezialisierung. Sie wird auch als *is-a*-Relation bezeichnet, da ein erbender Objekttyp eine Spezialisierung seines Vaters ist (wobei neue Attribute hinzukommen können, aber nicht müssen).

Da jeder Objekttyp auch eine Art Spezialisierung seiner selbst ist, ist die Vererbung *reflexiv*:

(vii) $$\chi \angle \chi \quad | \; \forall \chi \in X$$

Sie ist *antisymmetrisch*, d.h. kein Objekttyp erbt von einem anderen Objekttyp, der wiederum von ihm erbt:

(viii) $$\chi \angle \chi' \wedge \chi' \angle \chi \Rightarrow \chi = \chi' \quad | \; \forall \chi, \chi' \in X$$

Sie ist zudem *transitiv*:

(ix) $$\chi \angle \chi' \wedge \chi' \angle \chi'' \Rightarrow \chi \angle \chi'' \quad | \; \forall \chi, \chi', \chi'' \in X$$

Damit ist die Vererbungsrelation eine Halbordnung.

6.4.2 Definition von Instanzen

Ein Objekt *o* ist eine Menge von Attribut-Wert-Paaren:

(x) $$o : \{(\alpha_i, v_i) \mid \alpha_i \in A\}$$

Dabei ist die Menge A_o definiert als die α, denen das Objekt *o* einen Wert zuordnet:

(xi) $$A_o = \{ \alpha \mid \exists (\alpha, v) \in o\}$$

Ein Objekt *o* ist *valide* bezüglich eines Attributsschemas κ, wenn alle Werte der Attribute vom richtigen Attributstyp sind:

(xii) $$valid(o,\kappa) ::= \forall (\alpha, v) \in o : v \text{ ist vom Attributstyp } \kappa(\alpha)$$

Ein Objekt *o* ist eine *Instanz* eines Objekttyps χ, wenn alle erforderlichen Attribute von χ in *o* auch einen Wert zugeordnet bekommen:

(xiii) $$\chi \ll o ::= A_{erf} \subseteq A_o \quad | \; (iii)$$

Damit kann ein Objekt auch Instanz mehrerer Objekttypen sein.

Ein Objekt *o* ist *konform* bezüglich eines Schemas λ, wenn es bezüglich des zugehörigen Attributschemas valide ist, und alle seine Attribute von einem Objekttyp des instanziierenden Schemas definiert werden:

(xiv) $compliant(o,\lambda) ::= valid(o,\kappa) \wedge \alpha \in A_o \exists \chi \in X: (\chi \ll o \wedge \alpha \in A_{erf} \cup A_{opt})$ | *(xii)(iii)(i)(x)(xiii)*

6.4.3 Definition des Standard-Schemas

Mit Hilfe der obigen Definitionen lässt sich nun ein Standard-Schema λ_s mit einer Objekttypenmenge X_s definieren, das einen ausgezeichneten Wurzel-Objekttyp χ_{root} enthält:

(xv) $\exists \chi_{root} \in X_s: \forall \chi \in X_s: \chi_{root} \angle \chi$ | $\lambda_s = <\kappa_s, X_s, \angle>$

Der Wurzel-Objekttyp χ_{root} erbt nur von sich selbst, hat also keine Eltern:

(xvi) $\chi \angle \chi_{root} \Rightarrow \chi = \chi_{root}$

Der Wurzel-Objekttyp χ_{root} hat den Namen *NexusObject* und definiert ein Attribut mit dem Namen *type*, das vom Attributtyp *string* ist:

(xvii) $\chi_{root} = <NexusObject, A_{erf,root}, A_{opt,root}>$

(xviii) $type \in A_{erf,root}$

(xix) $\kappa_s(type) = Type$ | $string \in B$

Das Attribut *type* erhält auf Instanzebene als Wert den Namen der zugehörigen Objekttypen. Da es von χ_{root} festgelegt wird, müssen alle Objekttypen und Objekte des Standard-Schemas dieses Attribut enthalten:

(xx) $\forall \chi \in X_s: \chi \ll o \Rightarrow (type, \delta) \in o$ | *(iii),(x)*

Das *type* Attribut schränkt die Instanz-Relation ein. Nun sind nur noch die Objekte Instanz eines Objekttyps, die dessen Namen auch als Attributwert von *type* definieren:

(xxi) $\chi \ll o ::= A_{erf} \subseteq A_o \wedge (type, \delta) \in o$ | *(iii),(xviii)*

6.4.4 Definition einer Schemaerweiterung

Ein Schema λ_e stellt dann eine Erweiterung zu einem Ursprungsschema λ_u dar, wenn es die Objekttypen X_u des Ursprungsschemas als Teilmenge enthält, alle eigenen Objekttypen X_e von Objekttypen X_u des Ursprungsschemas erben und das Attributsschema κ_u Teilmenge von κ_e ist:

(xxii)
$$extend(\lambda_u,\lambda_e) ::= X_u \subseteq X_e \wedge \kappa_u \subseteq \kappa_e \wedge \forall\chi\in(X_e - X_u)\ \exists\chi'\in X_u : \chi' \angle_e \chi \mid \lambda_u = <\kappa_u,X_u,\angle_u>,\ \lambda_e = <\kappa_e,X_e,\angle_e>$$

Erweitert also ein Schema das Standardschema λ_s, so erhält jeder Objekttyp des erweiterten Schemas auch das *type*-Attribut:

(xxiii)
$$extend(\lambda_s,\lambda_e) \Rightarrow \forall\chi\in X_e:\ \chi \ll o \Rightarrow (type, \delta) \in o$$

6.4.5 Definition von Operatoren

Es können nun auf Objekten und Objekt-Mengen Operatoren definiert werden, die eine Auswahl ermöglichen. Operatoren werden auf ein Objekt *o* angewendet und stellen fest, ob das Objekt Attribute α besitzt, die eine bestimmte Eigenschaft bezüglich eines Wertes *v* erfüllen. Diese Attribute werden als neues Objekt, also als Menge von Attribut-Wert-Paaren, zurückgegeben. Besitzt das Objekt keine Attribute, welche die Eigenschaft erfüllen, so wird die leere Menge zurückgegeben. Operatoren haben folgende Signatur:

(xxiv)
$$op: O \times A \times V \rightarrow O'$$

Generell gilt, das ein Operator die leere Menge liefert, sollte das Objekt kein Attribut α aufweisen:

(xxv)
$$op(o,\alpha,v) = \varnothing \ if\ \alpha \notin A_o \qquad \mid (x)$$

Operatoren können auch nur auf einer Teilmenge der Attributtypen B definiert sein; in diesem Fall ergeben sie die leere Menge, sollten sie auf ein Attribut angewendet werden, für dessen Attributstyp sie nicht definiert sind.

Eine Definition verschiedener Operatoren für die räumliche Anfragesprache dieser Arbeit findet sich in Tabelle 3. Durch die Semantik der Operatoren wird festgelegt, welche Attribut-Wert-Paare der Operator auswählt.

equal ist der Gleichheitsoperator und für alle Datentypen definiert.

Tabelle 3: Definierte Operatoren

Operator	Definiert für	Semantik
equal	B	*equal*(o,α,*v*) = {(α,*v′*) ∈ *o* | *v′*= *v*}
all	B	*all*(o,α,*v*) = {(α,*v*) ∈ *o* | ∀ *v*}
lesser	β∈ B : es gibt eine Ordnung < auf β	*lesser*(o,α,*v*) = {(α,*v′*) ∈ *o* | *v′* < *v*}
greater	β∈ B : es gibt eine Ordnung > auf β	*greater*(o,α,*v*) ={(α,*v′*) ∈ *o* | *v′* > *v*}
within	*point, polygon*	*within(*o,α,*v*) = {(α,*v′*) ∈ *o* | *v′* liegt in *v*}
overlap	*point, polygon*	*overlap(*o,α,*v*) = {(α,*v′*) ∈ *o* | *v′* überlappt sich mit *v*}
like	*string*	*like(*o,α,*v*) = {(α,*v′*) ∈ *o* | *v′* beginnt mit *v*}
equalType	*Type*	*equalType(*o,α,*v*) = {(α,*v′*) ∈ *o* | *v′* = *v* oder das Objekt ist eine Instanz eines Subtyps von α (s.u.)}

Der *all* Operator ignoriert den Vergleichswert und gibt alle Attribute mit dem Namen α zurück. Er wird später bei der Projektion benötigt, um Attributwerte unabhängig von ihrem Wert auszuwählen.

lesser und *greater* können Werte vergleichen, sofern auf diese eine Ordnung definiert ist.

within und *overlap* sind räumliche Operatoren. Sie sind nur auf den räumlichen Datentypen *point* und *polygon* definiert und vergleichen diese mit einer gegebenen geographischen Geometrie. Details zur Repräsentation räumlicher Daten finden sich in Kapitel 6.6.1.

Der Operator *like* ist auf Zeichenketten definiert und liefert die Attribute, deren Attributwert mit dem Vergleichswert beginnt.

Eine Besonderheit ist der Operator *equalType*. Wenn ein Schema λ gegeben ist, so erweitert er die Semantik von *equal*; es wird auch dann das Attribut *type* zurückgegeben, wenn das Objekt von einem Objekttyp instanziiert ist, der ein Subtyp von dem Objekttyp ist, der im Vergleichswert des Operators angegebenen ist:

(xxvi) $$equalType(o,type,v) = \{(\alpha,v') \in o \mid if\ \exists \chi,\chi' \in X: \chi \angle \chi' \text{«} o\} \qquad | (i)$$

In der Anfragesprache wird *equalType* als durch die Operator *equal* überladen: bei der Verarbeitung wird anhand des Attributs festgestellt, ob es sich um *type* handelt (und *equalType* angewendet wird) oder um ein anderes Attribut (und *equal* angewendet wird).

6.4.6 Definition von Selektion auf Objektmengen

Die in Kapitel 6.4.5 definierten Operatoren können nun verwendet werden, um Teile von Objektmengen O zu selektieren. Es werden genau die Objekte einer Menge ausgewählt, bei denen die Anwendung des Operators nicht die leere Menge liefert. Die Selektion σ bildet eine Objektmenge O, einen Operator op, ein Attribut α und einen Wert v auf eine neue Objektmenge O' ab:

(xxvii) $\sigma : O \times op \times A \times V \rightarrow O'$ | *(xxiv)*

oder auch: $O' = \sigma_{op,\alpha,v}\, O$

Dabei gilt:

(xxviii) $\sigma_{op,\alpha,v}\, O = \{\, o \in O \mid op(o,\alpha,v) \neq \varnothing \,\}$ | *(xxiv)*

Wie leicht zu sehen ist, ändert die Selektion nichts an der Schema-Konformität: wenn alle Objekte aus O schema-konform sind, so sind es auch alle Objekte aus O', da die Objekte entweder ganz oder gar nicht ausgewählt werden (also keine Attribute wegfallen können).

Selektionen können logisch miteinander verknüpft werden, um Boolesche Ausdrücke zu bilden. Dies kann einfach auf Mengenoperationen abgebildet werden.

6.4.7 Definition von Projektionen (Filter)

Mit Hilfe eines Filter-Operators können Teile von Objekten ausgewählt werden, in dem die Attribute spezifiziert werden, die das Ergebnisobjekt enthalten soll. Die Operatoren werden hierbei dazu verwendet, Attribute auszuwählen.

Ein Filter F ist definiert als eine Menge von Tripeln aus Operatoren op_i, Attributen α_i und Vergleichswerten v_i:

(xxix) $F = \{<op_i,\alpha_i,v_i>\}$ | *(xxiv)*

Dieser Filter kann nun auf eine Objektmenge *O* projiziert werden (π); das Ergebnis ist eine Vereinigung der Anwendung des Filters auf die einzelnen Objekte der Menge. Da bei der Anwendung des Filters auch leere Mengen auftreten können (Objekte, bei denen sich keine Attribute qualifizieren), wird vom Ergebnis die leere Menge abgezogen.

(xxx) $$\pi_F\, O = \{ \bigcup_{F,\, O} op_i(o_i, \alpha_i, v_i)\} - \varnothing \quad | \; (xxiv)(xxix)$$

Zu beachten ist hierbei, dass aufgrund der Mengeneigenschaft von π_F mehrere gleiche Objekte zu einem zusammengeführt werden (bzw. nur ein Exemplar erhalten bleibt). Dies ist eine gewollte Eigenschaft, da so bei der Vereinigung mehrerer lokaler Umgebungsmodelle Duplikate entfernt werden. Wünscht eine Anwendung diese Eigenschaft nicht, so kann sollte sie darauf verzichten, das Attribut, das die eindeutige Objekt-ID enthält, wegzufiltern. Damit werden die einzelnen Objekte auch nach der Projektion noch unterscheidbar sein.

Wenn die Anwendung nur einzelne Attribute wegfiltern möchte, so müsste sie alle anderen Attribute im Projektionsfilter angeben. Deswegen wird in der Anfragesprache zwischen dem *include*-Filter (mit der obigen Semantik) und dem *exclude*-Filter unterschieden; bei letzterem werden genau die Attribut-Wert-Paare zurückgegeben, die durch die Operatoren nicht ausgewählt werden:

(xxxi) $$\pi_{exclude,F}\, O = \{ \bigcup_{F,\, O} o_i \text{ - } op_i(o_i, \alpha_i, v_i)\} - \varnothing \quad | \; (xxiv)(xxix)$$

6.4.8 *Multiple*- und *Merge*-Operatoren

Neben dem impliziten Zusammenführen gleicher Objekte bei der Projektion *(xxx)* können auch spezielle *merge*-Operatoren definiert werden, die solche Objekte zusammen führen, die zuvor von einer semantischen Analyse identifiziert werden. Dabei handelt es sich in der Regel um Mehrfachrepräsentationen von physischen Objekten, die in verschiedenen lokalen Umgebungsmodellen enthalten sind.

Eine einfache Möglichkeit, solche Mehrfachrepräsentationen zu erkennen, besteht über die Objekt-ID (Kapitel 6.2.2).

Hierzu wird für alle Objekten ein Objekt-ID definiert. Im technischen Schema definiert erst der Objekttyp **NexusDataObject** (der von **NexusObject** direkt erbt) den NOL, der die Objekt-ID enthält (Kapitel 6.6.1). Da jedoch alle Objekte des Umgebungsmodells von **NexusDataObject** erben, wird ein Attribut *id* in der formalen Darstellung der Einfachheit wegen dem Wurzeltyp χ_{root} zugeordnet.

(xxxii) $$id \in A,\ id \in A_{erf,root} \quad | \quad (xvi)$$

Damit kann nun ein Operator *multiple* definiert werden, der für zwei Objekte feststellt, ob sie Mehrfahrrepräsentationen sind:

(xxxiii) $$multiple : O \times O \rightarrow \{true, false\}$$

Für den Vergleich über Objekt-IDs ist der Operator so definiert, dass er dann *true* liefert, wenn beide Objekte ein Attribut *id* besitzen, das den gleichen Wert *v* hat:

(xxxiv) $$multiple(o,o') = true \text{ if } id \in A_o \cap A_{o'} \wedge \exists v : (id,v) \in o \cap o'$$

Weitere semantische Analysen können den Operator erweitern, so dass er auch in anderen Fällen *true* liefert (wenn z.B. zwei geographische Objekte so ähnlich sind, dass sie sich vermutlich um das gleiche Realweltobjekt handeln).

Der *merge*-Operator führt nun in einer Objektmenge *O* die Objekte *o* zusammen, bei denen der *Multiple*-Operator den Wert *true* liefert und bildet diese auf eine neue Objektmenge ab. Objekte, die nicht mehrfach repräsentiert sind, bleiben erhalten.

Dabei werden gleiche Attribut-Wert-Paare herausgefiltert (durch die Mengeneigenschaft gegeben):

(xxxv) $$merge: O \rightarrow O'$$

(xxxvi) $$merge\ O = \{o_i \cup o_j \in O \mid o_j \neq o_i \wedge multiple(o_i,o_j) = true\} \cup \{o_i \in O \mid \neg\exists o_j \neq o_i \in O : multiple(o_i,o_j) = true\}$$

6.4.9 Schemaanpassung von Objekten

Die Definition aus Kapitel 6.4.4 kann dazu verwendet werden, um einen Operator zu definieren, der ein Objekt eines erweiterten Schema λ_e in ein Objekt eines Ursprungsschema λ_u überführt (*upcast*):

(xxxvii) $$upcast : \lambda_u \times \lambda_e \times O \rightarrow O'$$

Der Operator *upcast* ist nur unter den folgenden zwei Vorbedingungen definiert:

(xxxviii) $$extend(\lambda_u,\lambda_e) = true \quad | \ (xxiii)$$

(xxxix) $$compliant(o,\lambda_e) = true \quad | \ (xiv)$$

Er bildet dann das Objekt *o* auf ein Objekt *o'* ab, das nur Attribute enthält, die von Objekttypen aus X_u definiert werden. Um möglichst viele Attribute zu erhalten, werden dazu die speziellesten passenden Objekttypen χ_{up} aus X_u verwendet, die vom Operator *up* bestimmt werden. Dass diese immer bestimmt werden können, folgt aus der Halbordnungseigenschaft der Vererbungsbeziehung $\angle$:

(xl) $$up\_t(\lambda_u,o) = \{\chi_{up} \in X_u \mid \chi_{up} \ll o \wedge \neg\exists \chi \neq \chi_{up} \in X_u : \chi_{up} \angle \chi \ll o\} \mid \lambda_u = <\kappa_u, X_u, \angle_u>$$

$up\_t(\lambda_u,o)$ liefert also eine Menge von Objekttypen X_{up}. Sei

(xli) $$A_{up} = A_{erf} \cup A_{opt} \ \forall \chi \in X_{up} \quad | \ (iii)$$

die Menge aller Attribute, die von einem Objekttypen aus X_{up} spezifiziert werden.

Der Operator *up* ist für ein Objekt *o* dann wie folgt definiert:

(xlii) $$up(\lambda_u,\lambda_e,o) = \{(\alpha,v) \in o \mid \alpha \in A_{up}\} \quad | \ (x)(xxxviii)(xxxix)(xlii)$$

Für eine Objektmenge *O* kann somit der Operator *upcast* definiert werden:

(xliii) $$upcast(\lambda_u,\lambda_e,O) = \{ \bigcup_{o \in O} up(\lambda_u,\lambda_e,o)\}$$

6.4.10 Anfrageverarbeitung

Mit diesen Grundlagen kann nun die Anfrageverarbeitung formal beschrieben werden.

Gegeben sei ein Standardschema λ_s *(v)*, ein Objektmenge *O* in einem Schema λ_e mit *extend*(λ_e,λ_s) = *true*, ein Selektionsausdruck *(xxviii)* und ein Filter *F (xxix).* Wir nehmen an, dass die anfragende Anwendung nur das Standardschema kennt: die erweiterten Objekte müssen also in Standard-Objekte umgewandelt werden.

Zunächst wird der Selektionsausdruck auf der Objektmenge ausgewertet. Das Ergebnis ist eine Objektmenge *O'*.

Auf diese wird nun erst der Filter *F,* dann der *upcast*-Operator und dann der *merge*-Operator angewendet:

(xliv) Ergebnismenge = *merge upcast*(λ_s,λ_e, π_F *O')*

Diese Reihenfolge ist sinnvoll, um eine effiziente Anfrageverarbeitung auf einer verteilten Objektmenge *O* zu ermöglichen. Die einzelnen Umgebungsmodellserver werten auf ihren lokalen Objektmengen nur *upcast*(λ_s,λ_e, π_F *O')* aus. Der *merge*-Operator wird erst auf Föderationsebene sinnvoll, die zunächst die einzelnen Ergebnismengen der Umgebungsmodellserver vereinigt und dann Mehrfachrepräsentationen erkennt und verschmilzt (Kapitel 5.3.1).

6.5 Schemadateien

Nachdem nun eine formale Grundlage der Modellierung gelegt wurde, wird im Folgenden die technische Umsetzung aufgezeigt.

Abbildung 11 zeigt den Zusammenhang zwischen den verschiedenen Schemadateien, die das Umgebungsmodell und seine Austauschsprachen beschreiben. In diesem Kapitel wird die Funktion der einzelnen Schemata beschrieben, in den folgenden deren Inhalt. Die kompletten Schemadefinitionen sind in [BDG+04] beschrieben und unter [AWS] zu finden.

Nexus Standard Attribute Types

Das Schema *NexusStandardAttributeTypes* (*nsat*) definiert die Basistypen des Umgebungsmodells (siehe Kapitel 6.6.1). Für Geometrien und Zeittypen importiert es die Schemadokumente von [GML]. Die Basistypen entsprechen den Attributtypen B, wie sie in Kapitel 6.4 definiert wurde.

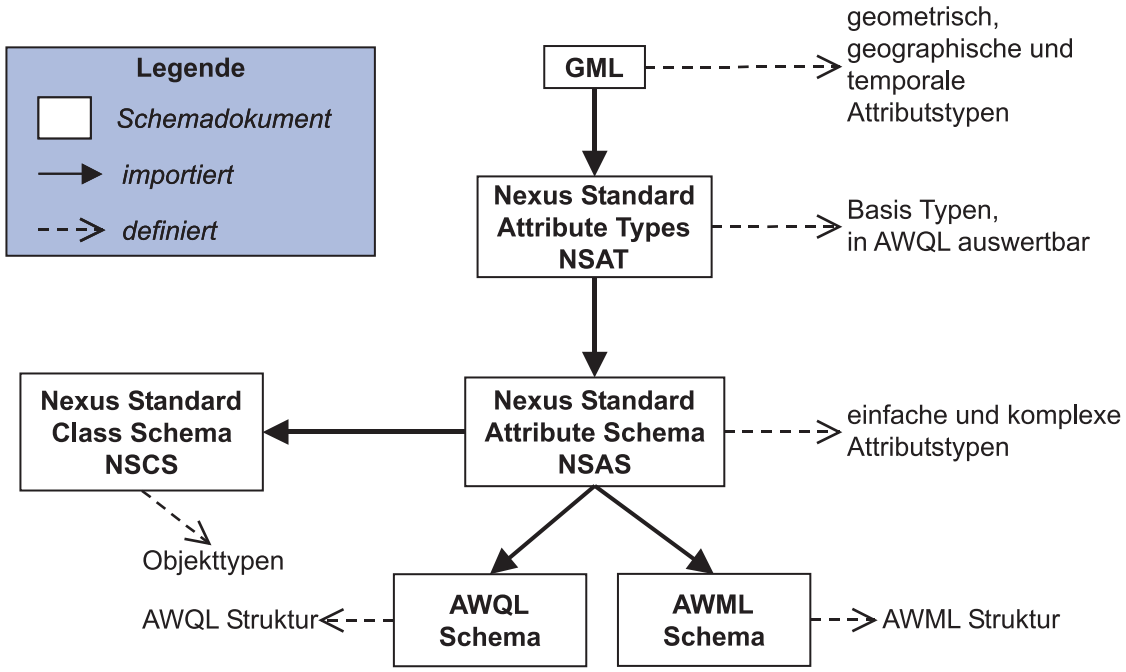

Abbildung 11: Schemadateien des Umgebungsmodells

Die Basistypen wirken sich direkt auf die Implementierung der Plattform und deren Verarbeitungskonzepte aus. Die Operatoren der Anfragesprache sind bezüglich der Basistypen definiert, und in den Plattformkomponenten muss Code zur Verfügung stehen, um Daten der Basistypen verarbeiten und darstellen zu können. Wenn also neue Basistypen hinzugefügt werden sollen, so hat das in der Regel weitreichende Auswirkungen auf die Implementierung (es sei denn, sie stellen nur Einschränkungen bestehender Basistypen dar).

Nexus Standard Attribute Schema

Das *NexusStandardAttributeSchema* (*nsas)* importiert die *NexusStandardAttributeTypes*, um damit einfache und komplexe (zusammengesetzte) Attributtypen zu definieren (siehe Kapitel 6.6.2). Es entspricht dem Attributschema κ, wie es in Kapitel 6.4 definiert wurde.

Grundsätzlich hätten die Basistypen auch innerhalb des Schemadokuments für die Attributtypen (NSAS) definiert werden können. Der Grund für die Trennung liegt in der Verarbeitung der Modelldaten:

- Bestimmte Basistypen bedingen bestimmte Operatoren bei der Anfrageverarbeitung. So sind z.B. für den räumlichen Basistyp *NexusPointType* Operatoren wie *within* oder *overlaps* definiert. Eine Erweiterung des NSAT kann also eine Erweiterung der Operatoren in AWQL erforderlich machen, sofern die neuen Typen sich nicht mit bestehenden Operatoren abprüfen lassen.
- Die Plattformkomponenten müssen in der Lage sein, die Daten der Basistypen zu verarbeiten und die Operatoren darauf anzuwenden. So ist z.B für den räumlichen Basistyp *NexusPointType* Programmcode vorhanden, der die verschiedenen Erscheinungsformen (WKT oder GML) erkennt, und daraus im Hauptspeicher eine Repräsentation erstellt, die von Anwendungen unabhängig von der AWML-Darstellung genutzt werden kann. Eine Erweiterung des NSAT kann also auch die Erweiterung verschiedener Plattformkomponenten nötig machen. Durch die Trennung der Schemata ist klar ersichtlich, ob eine Schemadatei eines erweiterten Schemas (siehe Kapitel 7.1) eine Implementierungserweiterung nach sich zieht oder nicht.

Nexus Standard Class Schema

Im *NexusStandardClassSchema* werden die Objekttypen definiert und ihren Attributen die Attributtypen aus dem *NexusStandardAttributSchema* zugeordnet (siehe Kapitel 6.7). Dies entspricht einem Schema λ aus Kapitel 6.4. Das Schema wird in der Plattform eingesetzt, um die Validität von Anfragen und Antworten zu prüfen. Wenn Ergebnisdokumente in einem der Anwendung unbekannten Schema zurückkommen, das eine Erweiterung des Standard-Schemas darstellt, so werden diese semantisch umgeformt (mit dem *upcast*-Operator, Kapitel 6.4.9 *(xliii)).*

AWML Schema

AWML (*Augmented World Modeling Language)* ist die Serialisierungssprache des Umgebungsmodells. Aus verschiedenen Gründen (die in Kapitel 6.8 erläutert werden) kann das *NexusStandardClassSchema* nicht das XML Schema für die Serialisierungssprache sein, und die Korrektheit der Objekttypen in AWML kann nicht direkt mit XML Technologie überprüft werden. Deswegen importiert das *AWML Schema* nur das *NexusStandardAttributSchema.* Die Plattform setzt das AWML Schema ein, um die grundsätzliche Validität von Ergebnisdokumenten zu prüfen.

AWQL Schema
AWQL (*Augmented World Query Language)* ist die Anfragesprache für das Umgebungsmodell. Ähnliches wie für das *AWML Schema* gilt für AWQL. Deswegen importiert das *AWQL Schema* nur das *NexusStandardAttributSchema* und nicht das *NexusStandardClassSchema.* Die Plattform setzt das AWQL Schema dazu ein, um die Validität von AWQL-Dokumenten zu prüfen.

6.6 Attributtypen

Ein Attributtyp beschreibt die Struktur und den Aufbau gleichartiger Attribute. Die Attributtypen sind zweistufig definiert: die Basistypen beschreiben, welchen Datentyp die Werte eines Attributs annehmen können (z.B. *NexusStringAttributType*). Das Attributschema legt fest, wie die Basistypen zu Attributtypen kombiniert werden. Einfache Attributtypen enthalten nur Werte eines Basistypen. Komplexe Attributtypen können Werte verschiedener Basistypen kombinieren.

6.6.1 Basistypen

Manche der Basistypen werden direkt auf Datentypen abgebildet, die in [XML Schema] definiert sind. Diese werden in der folgenden Beschreibung durch den Präfix xs: gekennzeichnet (z.B.: *xs:string*). Geometrische und Zeit-Typen wurden aus [GML] übernommen, einem umfangreichen Standard zur Repräsentation von Geodaten (siehe auch Kapitel 3.3.1). Diese werden durch den Präfix *gml:* gekennzeichnet. In Abbildung 12 sind Beispiele für verschiedene Definitionen in XML Schema dargestellt.

Das Umgebungsmodell unterstützt folgende Basistypen:

nsat:NexusBooleanType
nimmt den Wert ‘true’ oder ‘false’ an, entspricht *xs:boolean* (siehe Abbildung 12).

nsat:NexusByteType
kann Werte von -128 bis +127 annehmen, entspricht *xs:byte*

nsat:NexusDirectionType
gibt eine Richtung als Gradzahl an. Dabei bedeutet: 0=Norden, 90=Osten, 180=Süden und 270=Westen; entspricht *xs:double* mit *minInclusive* = 0 und *maxExclusive* = 360 (siehe Abbildung 12)

nsat:NexusFloatType
eine Fließkommazahl, entspricht *xs:float*

nsat:NexusGeometryType
ein Geometrie-Wert. Wie in Kapitel 6.2.7 beschreiben, fußt das Umgebungsmodell auf einem geographischen Lokationsmodell. Es werden zwei Standards unterstützt, Geometrien der *Simple Feature Specification* [SFS] anzugeben: WKT [WZG+04], das von vielen räumlichen Datenbanken verstanden wird, und *gml:_geometry* (siehe Abbildung 12).

nsat:NexusWktType
Der *NexusWktType* wird nicht direkt zur Definition von Attributtypen verwendet, sondern nur innerhalb der geometrischen Basistypen als Alternative zu GML. Wie in Abbildung 12 zu sehen, muss bei WKT ein Code für das Koordinatensystem extra angegeben werden. Bei GML ist dies durch den Standard selbst vorgegeben.

nsat:NexusIntegerType
ein numerischer Wert, entspricht *xs:integer*

nsat:NexusKindType
ein Aufzählungtyp, kann die Werte *virtual* oder *real* annehmen (siehe Abbildung 12).

nsat:NexusListType
eine Liste. Wird in der XML-Darstellung auf einen *xs:string* abgebildet, der die einzelnen Listenelemente getrennt durch Kommas enthält

nsat:NexusObjectLocatorType
Ein NexusObjectLocator (NOL) dient der Identifikation und der Auffindbarkeit eines Objekts. Er hat folgenden Aufbau:
nexus:<schema_id>:<endpoint>|<service>|<aaid>/<object_id>
Wenn die **<schema_id>** den Wert '*http*' hat, hat der **<endpoint>** folgenden Aufbau:
//<servername>:<port>/<appendix>
Die **<schema_id>** gibt an, mit welchem Protokoll das Objekt abzufragen ist. Bisher ist nur *http* [HTTP] spezifiziert und implementiert.
Der **<endpoint>** ist die URL des Umgebungsmodellservers, über den das Objekt zugreifbar ist.
Der **<service>** gibt an, über welchen Dienst das Objekt abfragbar ist.

Die **<aaid>** ist die weltweit eindeutige ID des lokalen Weltmodells (*Augmented Area),* zu dem das Objekt gehört.
<schema_id>, <endpoint>, <service> und **<aaid>** geben also gemeinsam den Speicherort des Objekts an.
Die **<object_id>** ist die ID des Objekts. Für sie gilt: die ID ist eindeutig innerhalb eines lokalen Weltmodells. Existieren in anderen lokalen Weltmodellen Objekte mit der gleichen ID, so repräsentieren diese das gleiche Realweltobjekt.
In der XML-Darstellung wird dieser Aufbau nur ansatzweise festgeschrieben: der Typ ist definiert als Einschränkung des *xs:string*-Typs auf das Pattern **nexus:.***.

```
<simpleType name="NexusBooleanType">
  <restriction base="boolean"/>
</simpleType>

<simpleType name="NexusDirectionType">
  <restriction base="double">
    <minInclusive value="0"/>
    <maxExclusive value="360"/>
  </restriction>
</simpleType>

<complexType name="NexusGeometryType">
  <choice>
    <element name="WKT" type="nsat:NexusWktType"/>
    <element ref="gml:_Geometry"/>
  </choice>
</complexType>

<complexType name="NexusWktType">
  <simpleContent>
    <extension base="string">
      <attribute name="srscode" type="nsat:NexusSrscodeType" use="required"/>
    </extension>
  </simpleContent>
</complexType>

<simpleType name="NexusKindType">
  <restriction base="string">
    <enumeration value="real"/>
    <enumeration value="virtual"/>
  </restriction>
</simpleType>

<complexType name="NexusPointType">
  <choice>
    <element name="WKT" type="nsat:NexusWktType"/>
    <element ref="gml:Point"/>
  </choice>
</complexType>
```

Abbildung 12: Schemadefinition von Nexus Basistypen

```
<complexType name="NexusBooleanAttributeType">
  <sequence>
    <element name="value" type="nsat:NexusBooleanType"/>
    <element ref="nsas:meta" minOccurs="0"/>
  </sequence>
</complexType>

<complexType name="NexusPointAttributeType">
  <sequence>
    <element name="value" type="nsat:NexusPointType"/>
    <element ref="nsas:meta" minOccurs="0"/>
  </sequence>
</complexType>
```

Abbildung 13: Schemadefinition einfacher Attributtypen

nsat:NexusPointType
Ein geographischer Punkt. Es werden zwei Standards unterstützt, um Punkte (*Points*) der *Simple Feature Specification* [SFS] anzugeben: **POINT** aus WKT [WZG+04] und *gml:Point* (siehe Abbildung 12).

nsat:NexusPercentageType
Ein Prozentwert – angegeben zwischen 0 und 1.

nsat:NexusSpeedType
Gibt eine Geschwindigkeit in Meter/Sekunde an. Wird durch *xs:double* repräsentiert

nsat:NexusStringType
Eine beliebige Zeichenkette, entspricht *xs:string*

nsat:NexusTimeType
Ein Zeitpunkt oder ein Zeitraum, entspricht *gml:_TimePrimitive*

nsat:NexusUriType
Eine *Uniform Resource Identifier*, entspricht *xs:anyUri* [URI]. Damit können z.B. beliebige Web-Ressourcen identifiziert werden.

6.6.2 Attributsschema

In diesem Kapitel werden nur Beispiele für die Definitionen gegeben, eine komplette Liste aller Attributtypen findet sich unter *NexusStandardAttributeSchema* im [AWS].

Zu jedem Attribut kann ein optionales Element <*meta*> definiert werden, mit dem zusätzliche Informationen zu einem Attribut (z.B. Genauigkeit, Autor oder Gültigkeitszeit) angegeben werden können. Die Möglichkeit, Metadaten auf Attributsebene zu modellieren, ist eine neuere Entwicklung des Umgebungsmodells, die noch nicht abschließend evaluiert ist. Deswegen werden in dieser Arbeit nur vorläufige Ergebnisse in Kapitel 7.3.1 beschrieben.

In Abbildung 13 wird als Beispiel die Definition der folgenden einfachen Attributstypen gezeigt:

nsas:NexusBooleanAttributeType
: Ein einfacher Attributtyp, der einen Wert vom Basistyp *NexusBooleanType* enthält.

nsas:NexusPointAttributeType
: Ein einfacher Attributtyp, der einen Wert vom Basistyp *NexusPointType* enthält.

Die weiteren einfachen Attributtypen sind analog definiert.

```
<complexType name="NexusAddressAttributeType">
  <annotation>
    <documentation>
      a complex Nexus Attribute Type that contains the street address of an entity
    </documentation>
  </annotation>
  <sequence>
    <element name="value">
      <complexType>
        <sequence>
          <element name="street" type="nsat:NexusStringType" minOccurs="0"/>
          <element name="number" type="nsat:NexusStringType" minOccurs="0"/>
          <element name="city" type="nsat:NexusStringType" minOccurs="0"/>
          <element name="zipCode" type="nsat:NexusIntegerType" minOccurs="0"/>
        </sequence>
      </complexType>
    </element>
    <element ref="nsas:meta" minOccurs="0"/>
  </sequence>
</complexType>
```

Abbildung 14: Schemadefinition eines komplexen Attributtyps

Interessanter sind die komplexen Attributtypen. Diese werden immer dann verwendet, wenn mehrere Eigenschaften eines Objekts unmittelbar zusammen gehören, z.B. die einzelnen Teile einer Adresse. Die Teile werden gruppiert. Dabei ergibt sich eine Baumstruktur, deren Blätter immer Attributteile sind, die durch einen Basistyp definiert werden.

Die Schachtelungstiefe dieser Baumstruktur wird durch das Attributsschema fest vorgegeben. Rekursive Strukturen sind nicht erlaubt. Dies ermöglicht es, aus der Baumstruktur eine flache Darstellung zu bilden, bei der die einzelnen Attribute einen Namen erhalten, der aus dem Pfad im Baum entsteht. Der Vorteil hiervon ist, dass die Daten damit immer noch leicht auf ein relationales oder objektorientiertes Datenmodell abbildbar sind.

Metadaten können bei komplexen Attributen nur für das gesamte Attribut, nicht für die einzelnen Teile vergeben werden.

nsas:NexusAddressAttributeType
: Ein komplexer Attributtyp, der vier Attribute enthält: *street*, *number*, *city* vom Basistyp *NexusStringType*, *zipcode* vom Basistyp *NexusIntegerType*. Die Schemadefinition findet sich in Abbildung 14.

nsas:NexusColorAttributeType
: Ein komplexer Attributtyp, der die verschiedenen Aspekte und Darstellungsmöglichkeiten einer Farbe bündelt. Er enthält sechs Attribute, die alle vom Basistyp *NexusStringType* sind. *name* ist ein vorgeschriebenes Attribut und gibt einen menschenlesbaren Namen der Farbe an. Alle weiteren Attribute sind optional: *secondname* ist ein zweiter Name, mit dem grenzwertige Farben durch eine zweite Einschätzung gekennzeichnet können (z.B. blau – grün). *lightness* ist die Helligkeit, *saturation* die Sättigung, *rbgcode* gibt einen Farbwert in RGB-Kodierung und *ralname* den Namen nach dem RAL Standard.

6.7 Objekttypen

In diesem Kapitel werden nun die wichtigsten Objekttypen des Umgebungsmodells vorgestellt, wie sie sich aus der Szenarien-Analyse und der Entwicklung von Anwendungen ergeben haben.

6.7.1 Zentrale Objekttypen

NexusObject bildet die Wurzel des Objekttypenschemas, alle Objekttypen sind direkte oder indirekte Kinder davon. Es definiert ein vorgeschriebenes Attribut **type**, das konstant für alle Objektinstanzen den Namen des Objekttyps als Zeichenkette enthält. Damit können Anwendungen für jedes Objekt feststellen, welche Objekttypen sie instanziieren und welche Attribute sie damit innerhalb des Objekts zu erwarten haben.

NexusDataObject ist das einzige direkte Kind von **NexusObject**. Es definiert den Nexus Object Locator (**nol**) als eindeutige ID und Lokator für das Objekt. Das **kind** (Art)-Attribut legt fest, ob es sich um ein virtuelles oder reales Objekt handelt. **name** und **description** sind optional und bieten die Möglichkeit, eine sehr kurze und eine längere Beschreibung des Objekts zu modellieren. Erst Instanzen von **NexusDataObject** und davon erbende Objekte sind tatsächliche Objekte des Umgebungsmodells.

Diese beiden Objekttypen wurden getrennt modelliert, um noch weitere Kinder von **NexusObject** zu ermöglichen, die keine **NexusDataObject**s sind. Diese sind dann zwar keine Objekte des Umgebungsmodells, können jedoch teilweise mit den hier entwickelten Konzepten verarbeitet werden. Ein Beispiel hierfür sind Relationen (siehe Kapitel 7.3.4).

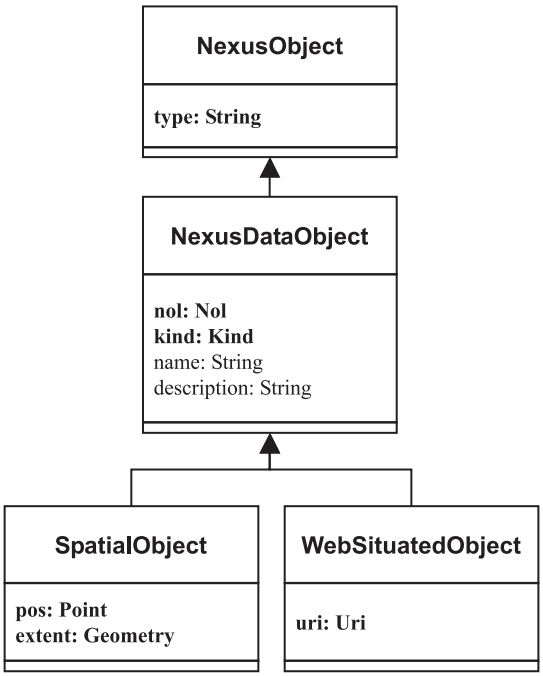

Abbildung 15: Zentrale Objekttypen

SpatialObject ist die Wurzel aller Objekte mit Ortsbezug. Es definiert zwei räumliche Attribute, **pos** (Position) und **extent** (Ausdehnung). Ein Ortsbezug ist für räumliche Objekte notwendig, deswegen sollte eines der beiden Attribute vorgeschrieben sein. Aber welches? Es gibt Objekttypen, die eher punktförmig modelliert werden können (z.B. eine Person mit ihrer Position), für andere ist es besser, eine Fläche zu modellieren, da die Position alleine keine hohe Aussagekraft hat (z.B. ein Straßenzug). Allerdings ist es möglich, zu einer Position eine Ausdehnung zu bestimmen (ein Polygon mit nur einem Punkt), und aus einer gegebenen Ausdehnung kann auch ein Punkt berechnet werden (z.B. der Schwerpunkt oder ein beliebiger Punkt innerhalb der Fläche). Für die Einfachheit des Umgebungsmodells (Anforderung 13) wäre es jedoch abträglich, wenn für manche räumliche Objekte die Position, für andere die Ausdehnung erforderlich wären. Es sollen sowohl Anfragen über die Position (z.B. die räumlich nächsten fünf Objekte) als auch über die Ausdehnung (z.B. alle, die sich mit meiner Position überschneiden) möglich sein. Dazu müssen beide Attribute bei allen räumlichen Objekten definiert sein. Deswegen sind sowohl **pos** als auch **extent** als erforderliche Attribute definiert.

WebSituatedObjects sind Objekte ohne Raumbezug, die über eine URI zugreifbar sind, z.B. eine Webseite. Diese können also nicht direkt über räumliche Anfragen ermittelt werden. Über ihre **nol** sind sie jedoch von räumlichen Objekten aus referenzierbar, z.B. über Relationen.

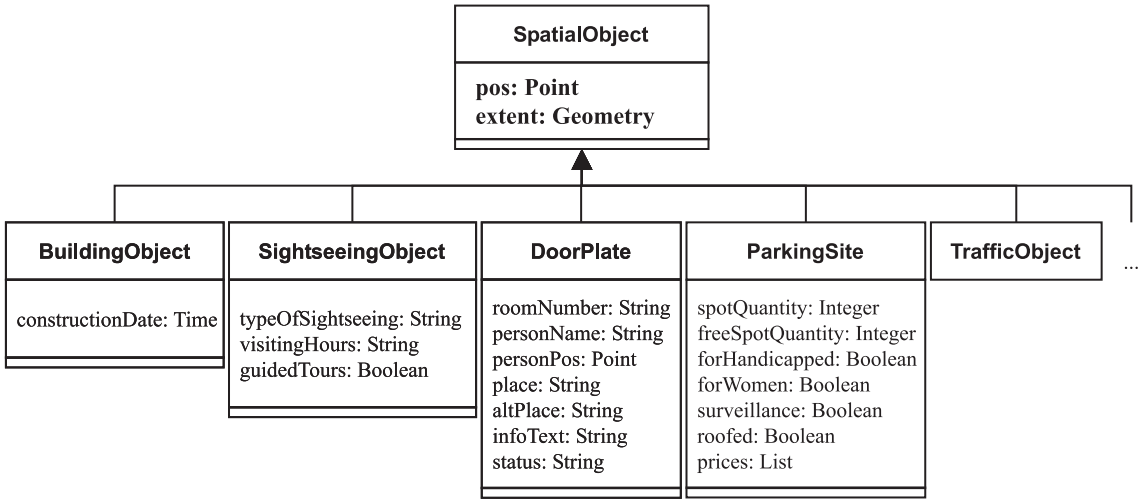

Abbildung 16: SpatialObject und seine Kinder

6.7.2 Landkarten: SpatialObject und seine Kinder

Eine wichtige Grundlage für viele ortsbezogene Anwendungen sind elektronische Landkarten, die aus räumlichen Objekten erzeugt werden können. Da es eine hohe Heterogenität in den tatsächlich erfassten Daten gibt, sind alle spezielleren Attribute von Landkartenobjekten optional. Die meisten Objekttypen des Umgebungsmodells fallen in diese Klasse. Abbildung 16 zeigt die wichtigsten Kinder von **SpatialObject** für diese Aufgabe: Gebäude(teile), Objekte mit touristischer Bedeutung, elektronische Türschilder, Parkplätze oder Verkehrsobjekte (**TrafficObject**), die in Kapitel 6.7.6 näher beschrieben werden. Darüber hinaus ist **SpatialObject** auch der Vater zahlreicher anderer Objekttypen wie **MobileObject** (Kapitel 6.7.3), **VIT** (Kapitel 6.7.4) oder **SensorObject** (Kapitel 6.7.5).

BuildingObject und seine Kinder sind in Abbildung 17 zu sehen. Hierbei handelt es sich um Objekte, die mit dem Aufbau von Gebäuden zu tun haben: Räume (**Room**), Stockwerke (**Level**), Wohnungen (**Apartment**) und ganze Gebäude (**Building**). Eine Wohnung kann sich über mehrere Stockwerke erstrecken und auch ein ganzes Gebäude ausfüllen. Unterhalb von **Apartment** finden sich die Objekttypen, die spezielle Funktionen von

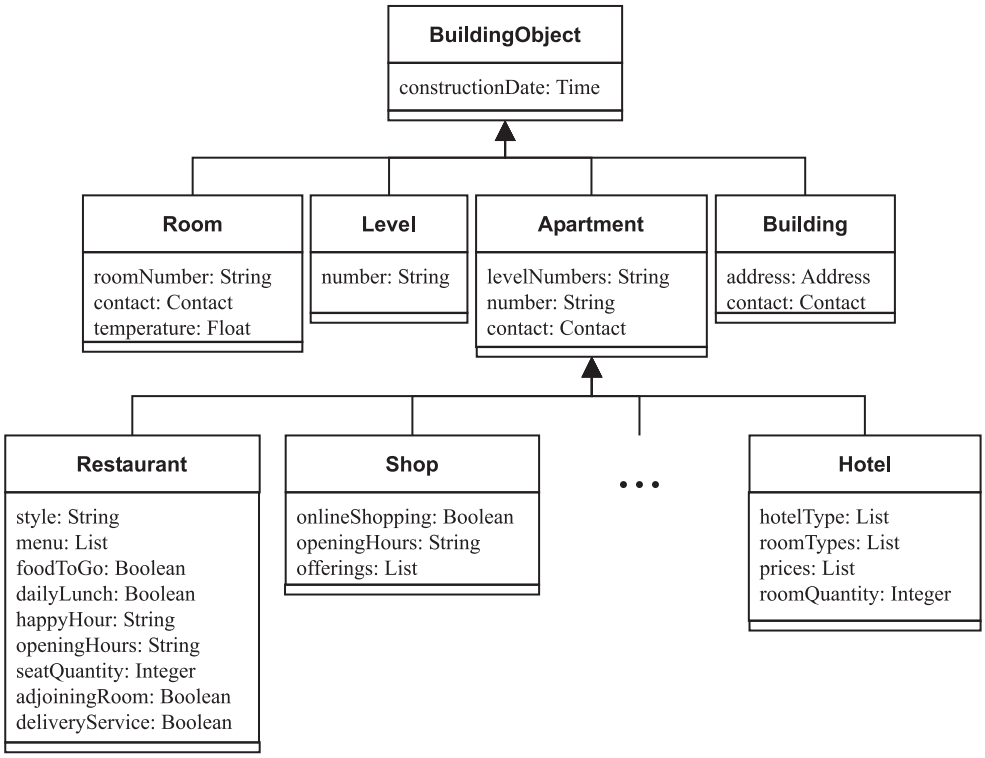

Abbildung 17: Gebäude: **BuildingObject** und seine Kinder

Gebäude (bzw. Teilen davon) definieren, wie z.B. **Restaurant, Shop** oder **Hotel**. Die verschiedenen **BuildingObject**s sind zum einen über ihre Geometrie implizit miteinander verbunden (da der **extent** des **Room**-Objekts innerhalb des **extent** der dazugehörigen **Level**, **Apartment** und **Building**-Objekte liegt, was über den **within**-Operator abgefragt werden kann). Zum anderen können, wenn für die Anwendung nötig oder vom Modellierer erwünscht, explizite **PartOf**-Relationen zwischen den Objekten definiert werden.

SightseeingObjects sind Objekte von besonderer touristischer Bedeutung, die über Öffnungszeiten oder spezielle Führungen verfügen können. An dieser Stelle bietet sich häufig eine Mehrfachvererbung an. Wie in Abbildung 18 zu sehen, kann von einer herkömmlichen Kirche (**Church**) und dem **SightseeingObject** eine **SpecialChurch** abgeleitet werden, die zwar keine eigenen Attribute besitzt, jedoch durch die gemeinsam vererbten Attribute seiner beiden Eltern definiert wird, also zusätzlich zu den Eigenschaften einer Kirche die eines touristisch interessanten Objekts erhält.

6.7.3 Mobile Objekte: MobileObject und seine Kinder

Mobile Objekte (Abbildung 19) sind dadurch gekennzeichnet, dass sie sich bewegen. Das führt dazu, dass ihre Position im Allgemeinen häufiger aktualisiert als abgefragt wird. Auf Serverseite ist deswegen eine gesonderte Behandlung sinnvoll [GBH+05]. Die Informationen, die zu einem mobilen Objekt definiert werden, stammen aus den Anforderun-

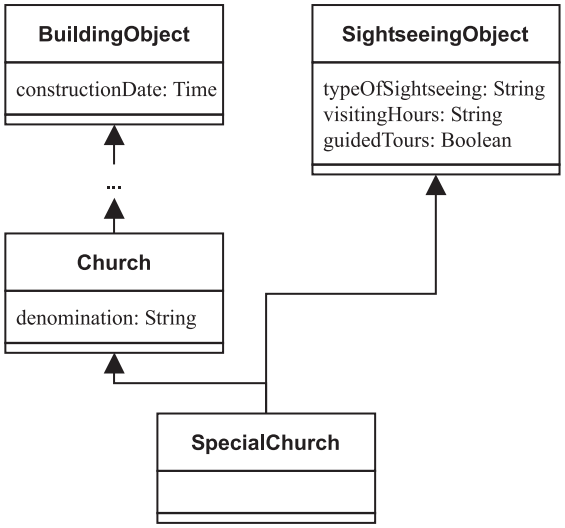

Abbildung 18: SightseeingObject: Beispiel für Mehrfachvererbung

gen aus [Leo03]: Durch Wissen über die Richtung und die Geschwindigkeit eines mobilen Objekts können beispielsweise sogenannte Koppelnavigations-Verfahren eingesetzt werden, um die Positionsdaten effizienter zu verwalten.

Kinder von **MobileObject** sind Personen, Tiere (z.B. für Anwendungen, welche die Position von Haustieren verfolgen) und Fahrzeuge. Wie bei **SightseeingObject** gibt es jedoch auch aus anderen Bereichen des Umgebungsmodells Objekttypen, die durch Mehrfachvererbung ebenfalls Kind von **MobileObject** sind, wie z.B. mobile Sensoren (**MobileIrBeaconReader**), die in Kapitel 6.7.5 beschrieben werden.

6.7.4 Virtuelle Objekte: Metaphern für die digitale Welt

Wie bereits erwähnt, besitzt jedes **NexusDataObject** ein Attribut **kind**, das angibt, ob das Objekt real ist (d.h. tatsächlich in der physischen Welt vorhanden) oder virtuell ist (d.h. nur über das Umgebungsmodell wahrnehmbar). Prinzipiell kann jedes Objekt virtuell sein, z.B. wenn es nur geplant ist. Virtuelle Objekte (Abbildung 20) werden im Umgebungsmo-

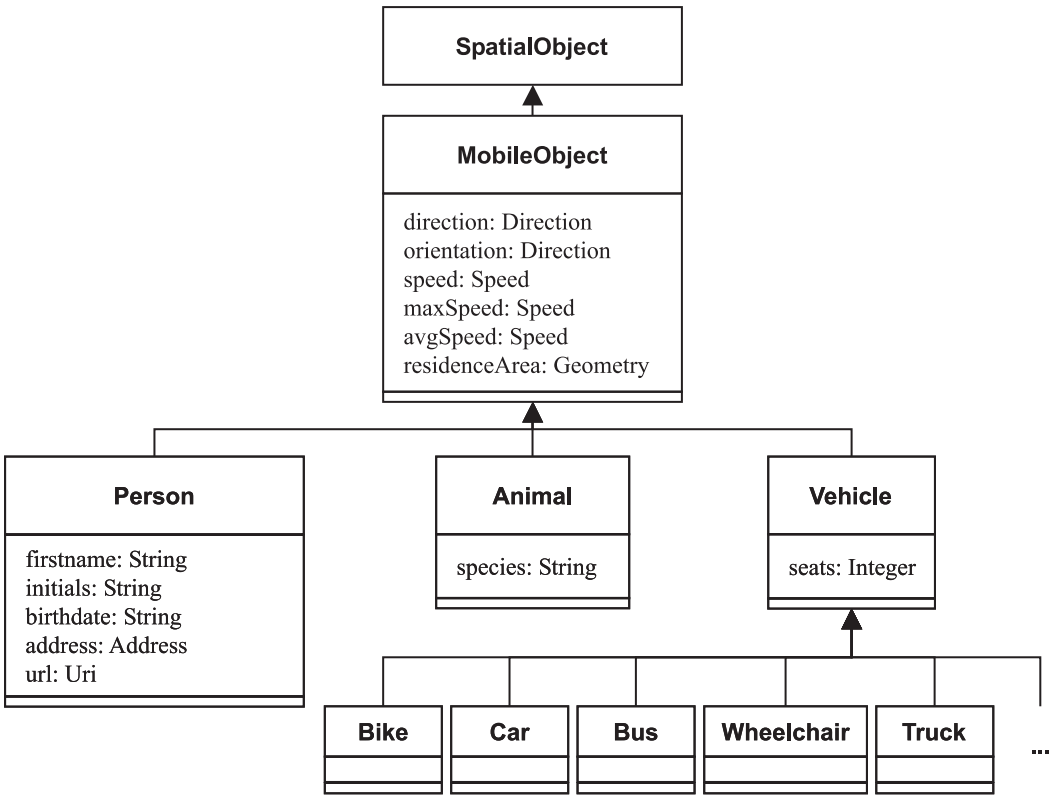

Abbildung 19: Mobile Objekte

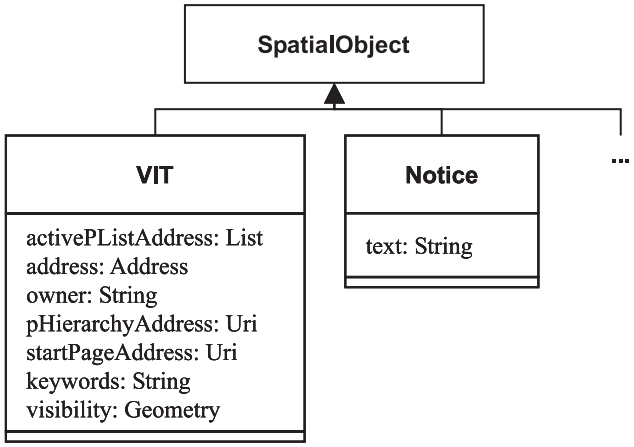

Abbildung 20: Virtuelle Objekte

dell auch dazu verwendet, um Metaphern für Informationsobjekte aus der digitalen Welt (z.B. Webseiten) zu modellieren, wie von [LKR99] erstmals vorgestellt. Eine virtuelle Litfaßsäule (*Virtual Information Tower,* kurz **VIT**) dient beispielsweise dazu, Webseiten an einem physischen Ort über das Umgebungsmodell zu „veröffentlichen". Sie hat eine physische Adresse, eine Liste von aktiven (direkt sichtbaren) Webseiten und einen Verweis auf eine Hierarchie weiterer Webseiten, die bei Bedarf von der Anwendung nachgeladen werden kann. Zudem verfügt sie über ein Sichtbarkeitsgebiet (**visibility**), in dem die verwiesenen Webseiten relevant sind – so wie auch eine reale Litfaßsäule nur in einem bestimmten Gebiet sichtbar ist. Während bei dem physischen Objekt dieses Gebiet jedoch von der Topographie bestimmt wird, kann es bei einer virtuellen Litfaßsäule frei nach der Relevanz der Inhalte bestimmt werden. Diese Unterschiede (Webseiten statt Papier-Poster, frei wählbare Visibilität) waren der Grund dafür, dass virtuelle Litfaßsäulen als eigener Objekttyp modelliert wurden (und nicht als etwaiges **InformationTowerObject**, bei dem das Attribut **kind** dann den Wert **virtual** hat). Dazu kommt, dass es aufgrund der Szenarien-Analyse und der bisherigen Anwendungen keine Notwendigkeit gab, reale Litfaßsäulen in das Modell aufzunehmen.

Der Objekttyp **Notice** dagegen kann sowohl als reales als auch als virtuelles Objekt auftauchen und bezeichnet eine ortsgebundene Notiz. Solange es jedoch noch kein elektronische Papier oder ähnliche Technologie gibt, mit der physische Notizen beobachtet und auch abgefragt werden kön-

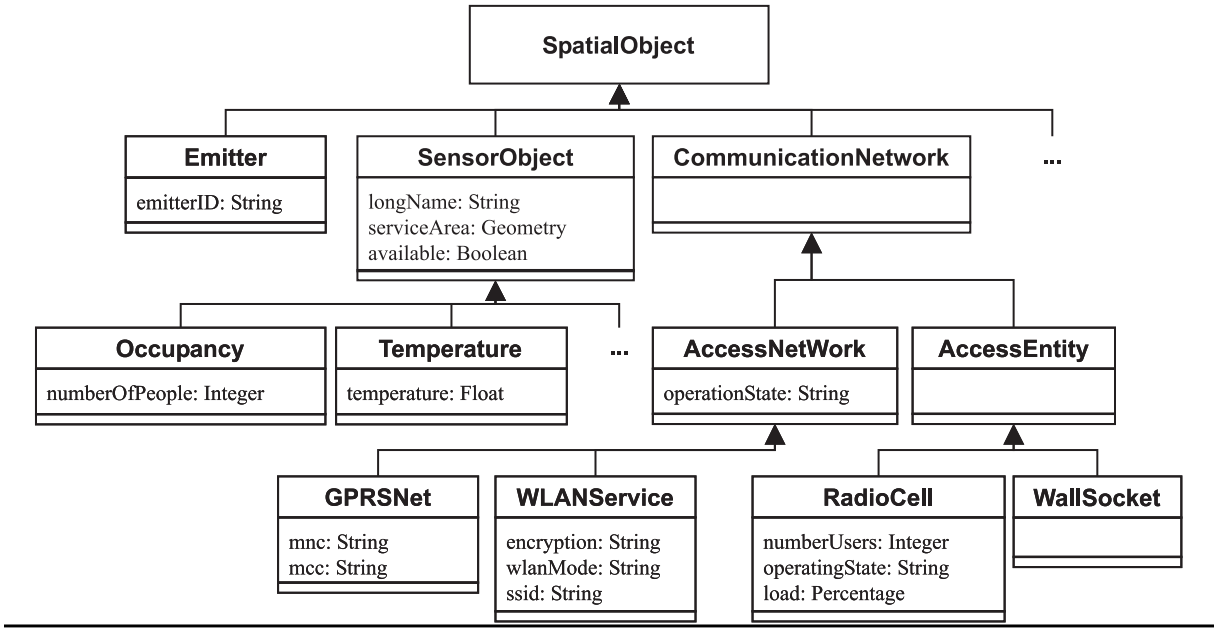

Abbildung 21: Technische Objekte

nen, werden auch Instanzen von **Notice** meist als rein virtuelle Objekte auftreten. Sie ermöglichen das Hinterlegen von ortsbasierten Informationen (Anforderung 2 (Ortsbasiertes Ablegen von Information)).

Ein weiteres rein virtuelles Objekt (*Virtual Task Container*) wird in einer Modellerweiterung definiert, die in Kapitel 8.5 vorgestellt wird.

6.7.5 Technische Objekte: Sensoren und Infrastruktur

Das Umgebungsmodell ermöglicht nicht nur das Modellieren von Objekten, die von Anwendungen direkt benötigt werden. Es können auch sogenannte technische Objekte dort abgelegt werden, welche Teil der Infrastruktur sind. Dadurch kann eine effizientere Modellverwaltung implementiert werden oder es können Anwendungen entwickelt werden, die beispielsweise technischem Personal bei der Wartung von kontextbezogenen Einrichtungen unterstützen. Einen Überblick über technische Objekttypen gibt Abbildung 21.

Emitter sind Objekte, die eine ID ausstrahlen, wie z.B. eine Infrarot-Bake. Diese können als ein raum-genaues Innenraum-Lokalisierungssystem verwendet werden, da Infrarot nur eine sehr begrenzte Reichweite hat. Wenn ein mobiles Endgerät also die ID eines Emitters empfängt, kann er über die im Umgebungsmodell gespeicherte Position des Emitters eine Näherung der eigenen Position herausfinden.

Unter **SensorObject** finden sich verschiedene Arten von Sensoren, z.B. für die Belegung eines Raums oder die Temperatur. Es gibt zwei Möglichkeiten, wie die gemessenen Werte eines Sensors im Umgebungsmodell dargestellt werden können. Zum einen können ein oder mehrere Sensoren dafür verantwortlich sein, dass ein bestimmtes Attribut eines anderen Objekts (z.B. die Temperatur eines Raumes in Abbildung 17) aktuell gehalten wird. Zum anderen kann der Sensor selbst als eigenständiges Objekt Teil des Umgebungsmodells sein (wie hier in Abbildung 21). Dadurch werden jeweils andere Anwendungen ermöglicht: eine Klimaanlagensteuerung wäre an der Temperatur in bestimmten Räumen interessiert, während eine Wartungsanwendung direkt den Sensor selbst finden möchte. Die explizite Modellierung von Sensoren und anderer technischer Objekte im Modell ermöglicht zudem auch die Offenlegung der ansonsten „versteckten“ Technik. Dies kann die Akzeptabilität von kontextbezogenen Anwendungen erhöhen, wenn der Benutzer jederzeit anfragen kann, welche Sensoren sich in dem Raum befinden, in dem er sich gerade aufhält.

Objekte vom Typ **CommunicationNetwork** stellen die Netzinfrastruktur dar, die Anwendungen zur Verfügung steht. Dabei wird unterschieden zwischen **AccessNetwork** – ein Zugangsnetz, das ein logischer Zusammenschluss verschiedener physischer Netzwerke sein kann – und **AccessEntity**, das einen konkreten Zugangspunkt in der physischen Welt beschreibt, wie z.B. eine Funkzelle oder eine Netzwerksteckdose. Durch die Modellierung der Netzinfrastruktur werden Dienste möglich, die dafür sorgen, dass ein mobiles Endgerät immer über den günstigsten oder schnellsten verfügbaren Zugangspunkt kommuniziert [LG04].

6.7.6 Verkehr und Navigation

Eine wichtige Anwendung für das Umgebungsmodell ist die Routenplanung oder auch Navigation, sei es zu Fuß, mit dem Auto oder mit öffentlichem Nahverkehr. Bisherige Systeme sind meist auf ein Verkehrsmittel beschränkt und lösen diese Aufgabe durch die Verwaltung spezieller Datensätze, die sowohl geographische Daten (konkrete Straßenverläufe) wie auch Navigationsdaten (z.B. Abbiegeverbote, geschätzte Fahrdauer) enthalten, die gemeinsam einen Graphen mit gewichteten Kanten bilden, auf dem dann eine kürzeste Route berechnet werden kann.

Durch den Ansatz des umfassenden Umgebungsmodells können nun Informationen verschiedener Verkehrsnetze (Fußgänger, Auto,...) verbunden werden. Um dies zu ermöglichen, wurden im Umgebungsmodell topologische von geographischen Daten getrennt. Für eine konkrete Navigationsanfrage wird zunächst ein Graph aus den topologischen Umgebungsmodelldaten abgeleitet, auf dem die Routenberechnung stattfindet. Anschließend kann das Ergebnis über Relationen mit den geographischen Daten verknüpft werden, um es beispielsweise in eine Karte einzuzeichnen.

Abbildung 22 zeigt zunächst geographische Verkehrsobjekte, beispielartig für den Straßenverkehr: Straßen bestehen aus **RoadElement**s, die entweder zu **SimpleRoad**s zusammengesetzt werden können. **ComplexRoads** bestehen aus mehreren **SimpleRoads**. Straßenkreuzungen bestehen aus **JunctionElement**s, die zu **ComplexJunction**s zusammengesetzt werden.

Die konkreten Objekttypen für diesen Anwendungsbereich wurden durch die Abbildung von bestehenden Navigationsdatenformaten auf eine einheitliche Modellierung gebildet [VGH+02]. In Tabelle 4 ist zu sehen, wie sich die Objekte der beiden gebräuchlichen Straßenverkehrsformate AKTIS und GDF auf die Objekttypen des Umgebungsmodells abbilden.

Tabelle 4: Zuordnung GDF, ATKIS und Umgebungsmodell (nach [VGH+02])

<table>
<tr><th>GDF</th><th>Relation</th><th>Umgebungs modell</th><th>Relation</th><th>AKTIS</th></tr>
<tr><td>Junction</td><td>1:1</td><td>Junction Element</td><td>2:1</td><td rowspan="2">Object Part</td></tr>
<tr><td rowspan="2">Road Element</td><td>1:1</td><td>Road Element</td><td>1:1</td></tr>
<tr><td>n:1</td><td rowspan="2">Simple Road</td><td>1:1</td><td>Object</td></tr>
<tr><td rowspan="2">Road</td><td>n:1</td><td></td><td></td></tr>
<tr><td>n:1</td><td>Complex Road</td><td>1:1</td><td>Complex Object</td></tr>
<tr><td>Intersection</td><td>1:1</td><td>Complex Junction</td><td></td><td></td></tr>
</table>

Die topologischen Objekttypen werden in Abbildung 23 dargestellt: es wird unterschieden zwischen **NavigationNode**, **Restriction** und **NavigationEdge**. **NavigationNodes** verbinden zwei **NavigationEdge**s, beide können durch eine **Restriction** eingeschränkt sein. Ein **NavigationNode** repräsentiert im Falle des Straßenverkehrs einen Straßenabschnitt und seine beiden abschließenden Kreuzungen. Die **NavigationEdge** verbindet zwei solcher Straßenabschnitte miteinander. Diese Modellierung entspricht nicht der naiven Abbildung eines Straßennetzes in einen Graphen, bei dem die Knoten die Kreuzungen und die Kanten die Straßen sind. Sie wurde jedoch notwendig, um komplexe Navigations-Informationen (z.B. Abbiegeverbote, Beschränkungen für bestimmte Fahrzeugtypen, etc.) abbilden zu können [VGH+02].

Aus der im Modell enthaltenen Information kann eine Anwendung oder ein Navigationsmehrwertdienst nun einen konkreten Navigationsgraphen bilden, der nur noch die Kanten enthält, die für die angestrebte Fortbewegungsart benutzbar sind.

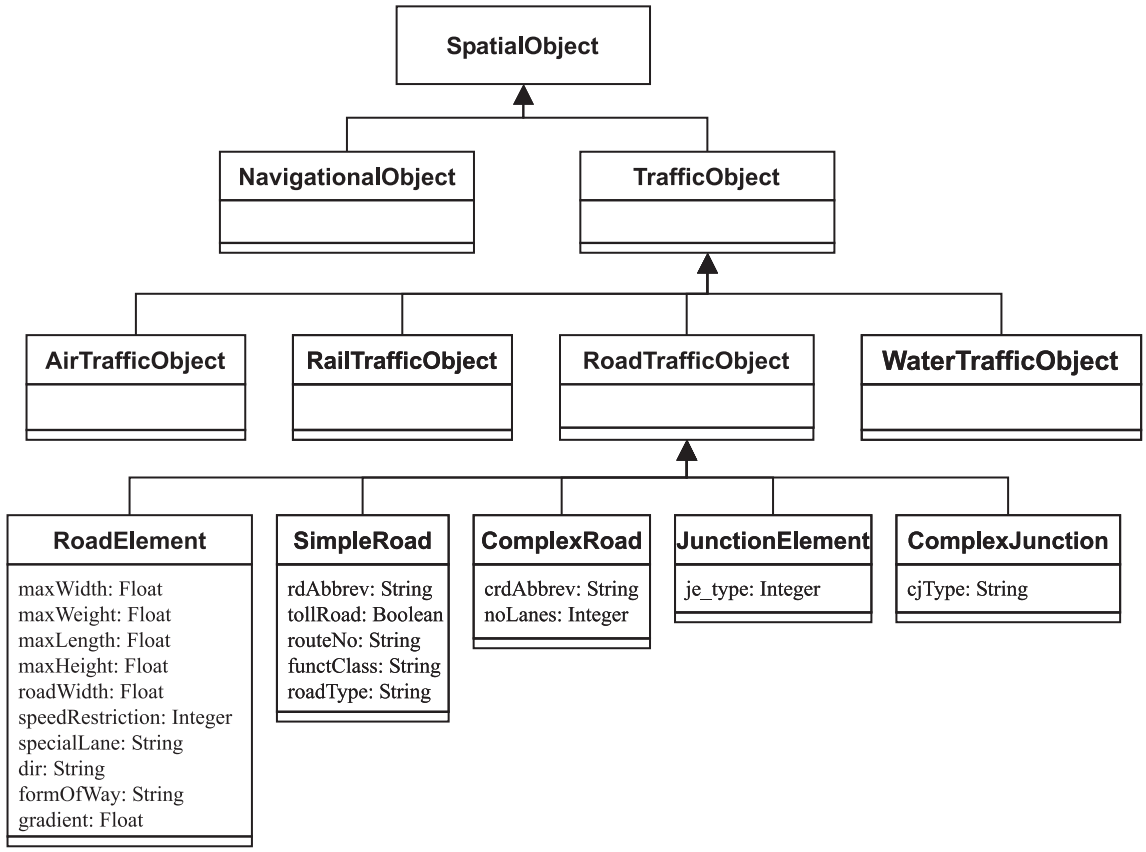

Abbildung 22: Verkehrsobjekte

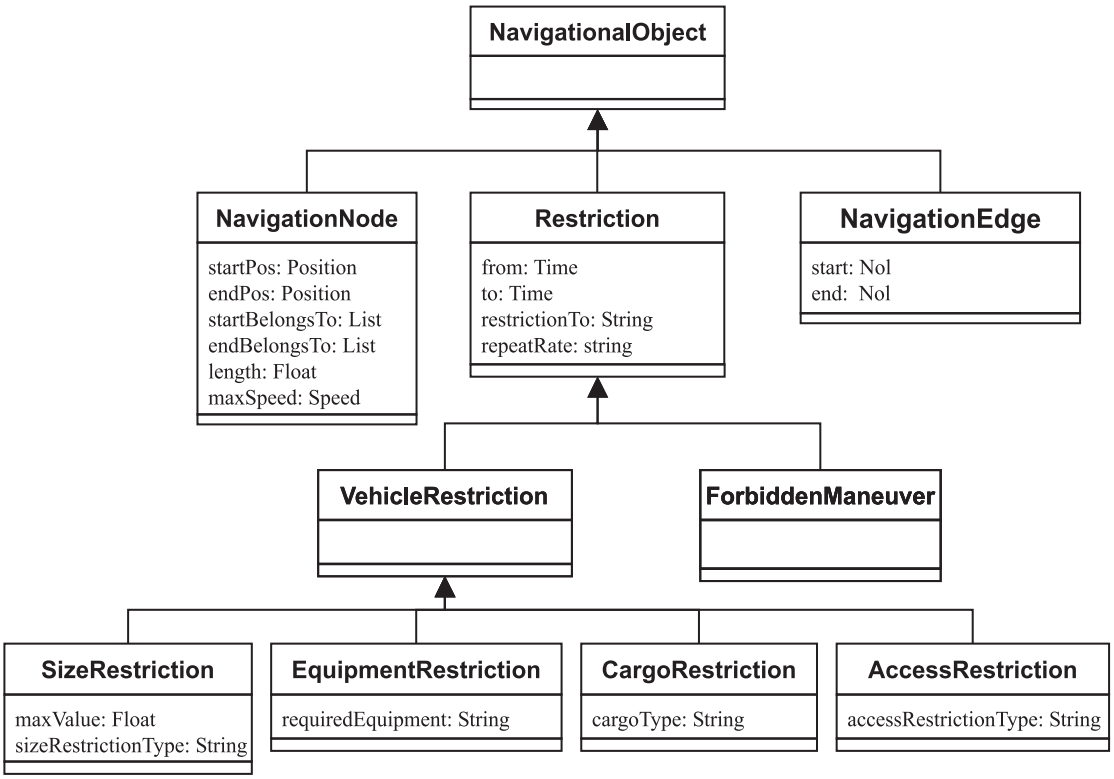

Abbildung 23: Navigationsobjekte

6.8 Serialisierung von Objekten (AWML)

Um Umgebungsmodelldaten zwischen Komponenten der Nexus-Plattform auszutauschen, wurde eine Sprache entwickelt, die Objekte des Modells serialisiert: die *Augmented World Modeling Language*, kurz *AWML*.

Im Folgenden werden nun die wichtigsten Eigenschaften und Entwurfsziele von AWML kurz vorgestellt. Die Sprache wird in der zum Zeitpunkt der Arbeit aktuellen Version 2.0 in [BDG+04] spezifiziert.

- AWML verwendet die *XML-Syntax* [XML]. Damit ist sie textbasiert und prinzipiell menschenles- und schreibbar, auch wenn dies von keinem Benutzer zu verlangen ist. Für Entwickler ist es jedoch unter Umständen nützlich, die Daten direkt mit einem Texteditor lesen und bearbeiten zu können. Ein Nachteil von XML ist sicherlich die „Geschwätzigkeit“: durch die Textbasierung mit Auszeichnung der Daten durch zusätzliche Tags haben Daten in XML-Formaten in der Regel ein wesentlich höheres Volumen als in einem kompakteren Textformat (z. B. mit sparsamerer Auszeichnung) oder gar Binärformat.

Der Vorteil von XML Formaten liegt in der Verfügbarkeit von zahlreichen Werkzeugen und Bibliotheken für verschiedene Systemplattformen. Damit wird ein hohes Maß an Interoperabilität erreicht. Zudem lässt sich das Umgebungsmodell mit seinen besonderen Eigenschaften gut mit XML darstellen, wie sich in den folgenden Punkten zeigen lässt, was der Erfüllung der Anforderung 13 (Einfachheit des Umgebungsmodells) zugute kommt. Sollte sich bei der konkreten Implementierung der Nexus-Plattform jedoch ein Konflikt mit der Anforderung 11 (Effiziente Verwaltung des Umgebungsmodells) ergeben, wäre es einfach möglich, die XML-Dokumente zu komprimieren, oder zwischen Komponenten ein kompakteres Format zu spezifizieren, das in seiner Semantik jedoch der XML-Darstellung entspricht.

- Die *Objektbasierung* des Umgebungsmodells spiegelt sich auch im Aufbau von AWML-Dokumenten wider. Die einzelnen Objekte werden durch XML Elemente dargestellt, welche die Attribute umschließen. Die Unabhängigkeit der Objekte bleibt erhalten: es ist jederzeit möglich, Objekte aus einem Dokument zu entfernen oder neue hinzuzufügen, ohne dass die anderen Objekte oder die Gültigkeit des Gesamtdokuments davon betroffen wären. Damit lassen sich beliebige Weltausschnitte darstellen, wie in Kapitel 6.2.1 gefordert.
- AWML verwendet sogenannte *generische Objekte*, um die Objekte des Umgebungsmodells darzustellen. Das bedeutet, dass die Objekt-Elemente alle den gleichen Tagnamen haben, anstatt beispielsweise den Objekttyp dort zu kodieren (**<nexusobject>** statt **<building>**). Der Objekttyp wird als Attribut **<type>** dargestellt und unterscheidet sich syntaktisch nicht von anderen Attributen. Da das Umgebungsmodell Objekte mit mehreren Objekttypen erlaubt, können in einem **<nexusobject>**-Element auch mehrere **<type>**-Elemente auftreten. Ohne generische Attribute sind Mehrfachtypen nicht darstellbar (oder nur durch eine wesentlich komplexere Darstellung über Referenzen), da ein XML-Element immer nur einen eindeutigen Namen haben kann.
- Die *optionalen Attribute* des Umgebungsmodells lassen sich in XML problemlos darstellen, da für jedes Element eine Kardinalität bestimmt werden kann, mit der es auftreten kann, die auch 0 sein kann.
- Die Struktur eines AWML-Dokuments ist die eines flachen Baums (siehe Abbildung 24): Die Wurzel bildet das **<awml>**-Dokument, die Objekte sind Zweige der ersten Ebene. An den Objekten hängen die Elemente, welche die Attribute darstellen und die aus ihrem Wert-

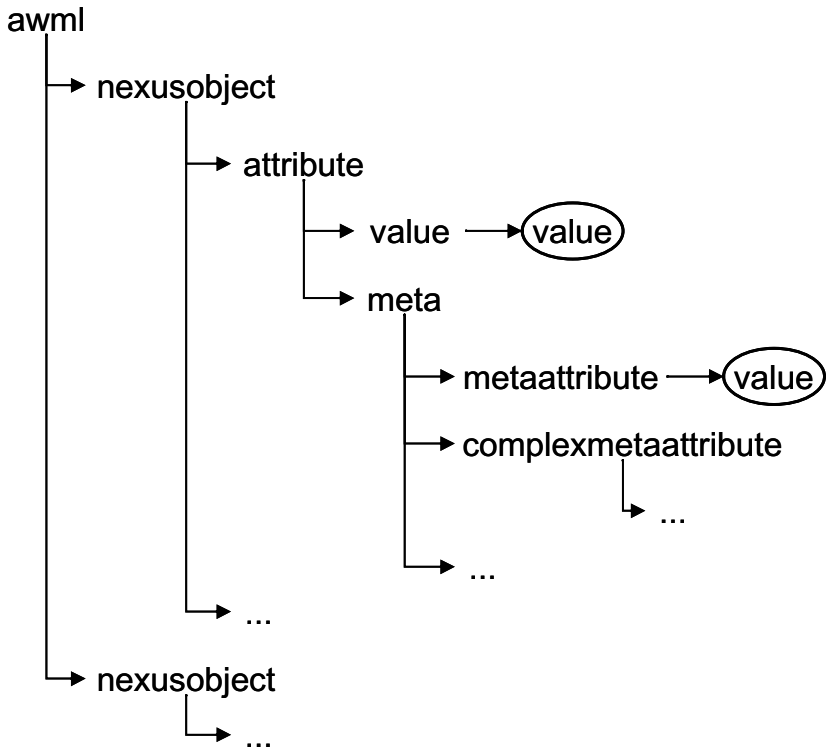

Abbildung 24: Struktur eines AWML-Dokuments (aus [BDG+04])

und ihrem Metadaten-Knoten bestehen (siehe dazu auch Kapitel 7.3.1). Diese Knoten können bei komplexen Attributen noch weitere hierarchische Strukturen aufbauen. Die Tiefe des Baumes ist prinzipiell unbegrenzt, allerdings ist sie vorab bekannt, da die komplexen Attributtypen im Attributsschema definiert werden (siehe Kapitel 6.6.2). Dadurch kann ein AWML-Dokument einfacher vom hierachischen XML-Modell auf ein flaches Relationen-Modell abgebildet werden: die Hierarchie der Attribute wird auf „flache“ Attributsnamen abgebildet, die aus dem Attributsschema gebildet werden können; aus **<address><street>** würde dann z.B. eine Tabellenspalte mit dem Namen **address.street.** Der konkrete Wert eines Attributs steht innerhalb des **<value>**-Elements, der eines Metaattributs im Element des jeweiligen Namens.

6.9 Anfragen an das Umgebungsmodell (AWQL)

Um Daten aus dem Umgebungsmodell abzufragen, wird eine Anfragesprache benötigt, die räumliche Operatoren zulässt, um die Anforderung 1 (Ortsbasierter Zugriff auf Information) zu erfüllen.

Dabei gibt es prinzipiell zwei Möglichkeiten: die Verwendung einer bestehenden oder die Entwicklung einer eigenen Anfragesprache.

Die Analyse der Anwendungen ergab, dass für die meisten Anwendungsfälle einfache Selektionen ausreichend sind, und auf komplexere Anfragekonstrukte wie *(Spatial) Joins* verzichtet werden kann. Auch die Anforderung 11 (Effiziente Verwaltung des Umgebungsmodells) erfordert eine Sprache, bei der auf Föderationsebene eine Vorverarbeitung der Anfrage möglich ist, um die Anzahl der Anfragen an die Umgebungsmodellserver möglichst gering zu halten.

Aus diesen Gründen schied SQL [SQL99] aus. Im SQL99 Standard stehen zwar räumliche Operatoren und Datentypen zur Verfügung, aber die Mächtigkeit der Anfragesprache ist zu groß, um sie noch verteilt effizient verarbeiten zu können. Auf Ebene der Umgebungsmodellserver kommt SQL99 jedoch zum Einsatz, um lokale Anfragen an räumliche Datenbanken zu bearbeiten.

Da in der Nexus-Plattform XML-Technologie zum Datenaustausch eingesetzt wird, wurden auch XML-Anfragesprachen wie *XQuery* [XQuery] oder *Quilt* [CRF00] auf ihre Eignung untersucht. Diese Sprachen sind für die Auswahl von Daten aus XML-Dokumenten gedacht. Sie bieten Funktionen, um aus dem hierarchischen Datenmodell eines XML-Dokuments Teilbäume und Blätter zu extrahieren. Zum Zeitpunkt der Entscheidung, welche Anfragesprache verwendet werden soll, gab es keine XML-Anfragesprachen, die räumliche Operatoren unterstützt hätten. Außerdem erfordert z.B. *XQuery* die Angabe einer Datenquelle (ein Dokument, spezifiziert durch eine URI), so dass keine Lokationstransparenz für Anwendungen vorhanden ist.

Deswegen wurde für die Nexus-Plattform eine eigene Anfragesprache entwickelt, die auf die Bedürfnisse der hier unterstützten Anwendungen zugeschnitten ist: die *Augmented World Query Language*, kurz *AWQL.*

AWQL ist eine einfache räumliche Anfragesprache, welche die Selektion von Objekten aufgrund von räumlichen und anderen Operatoren ermöglicht. Das gewünschte Ergebnis wird dabei in zwei Teilen spezifiziert: in der Objektselektion werden Operatoren angegeben, welche diejenigen Objekte auswählen, an denen die Anwendung interessiert ist (in SQL wäre dies die *WHERE*-Klausel). Im Attributsfilter werden dann die Attribute ausgewählt, welche die Anwendung im Ergebnis erwartet (SQL:

SELECT-Klausel). Die Angabe der Datenquelle (entspräche der *FROM*-Klausel von SQL) ist optional; im Normalfall stellt die Anwendung die Anfrage an das gesamte föderierte Umgebungsmodell, und die Föderation ist dafür verantwortlich, geeignete Datenquellen auszuwählen.

Neben der Selektion können mit AWQL auch Daten manipuliert (geändert oder gelöscht) werden.

6.9.1 Objektselektion

Die Objektselektion wird im **<restriction>-**Element der Anfrage spezifiziert. Diese ist ein boolescher Ausdruck, dessen Operatoren folgendem Muster folgen:

```
<operator>
  <target>Pfad zu einem Blatt im AWML-Baum</target>
  <referenceValue>Vergleichswert</referenceValue>
</operator>
```

Die Operatoren entsprechen dabei den Operatoren aus der formalen Darstellung in Kapitel 6.4. An der Stelle von **<operator>** kann dabei einer der folgenden Operatoren stehen (vgl. auch Tabelle 3):

<equal>
: Das **<target>** ist gleich dem **<referenceValue>**. Typ-Werte sind gleich, wenn sie entweder exakt übereinstimmen oder wenn einer der Typen vom anderen Typen direkt oder transitiv erbt.

<like>
: Für *string*-Vergleiche: der **<referenceValue>** ist ein *Substring* des **<target>**.

<greater> (<less>)
: Für numerische Werte: das **<target>** ist größer (bzw. kleiner) als der **<referenceValue>**

<within>
: Für räumliche Werte: das **<target>** liegt vollständig innerhalb des **<referenceValue>**

<intersects>
: Das **<target>** überschneidet sich mit dem **<referenceValue>**

Darüberhinaus stehen die booleschen Ausdrücke **<and>, <or>** und **<not>** in ihrer üblichen Semantik zur Verfügung, mit denen die Operatoren verknüpft werden können.

Ein Objekt wird dann ausgewählt, wenn eines seiner Attribute mit der **<restriction>** übereinstimmt.

6.9.2 Attributsfilter

Eine Anwendung kann über den Attributsfilter bestimmen, an welchen Attributen der durch die Selektion ausgewählten Objekte sie interessiert ist (Projektion). Dafür stehen zwei Ausdrücke zur Verfügung, von denen nur einer in der AWQL-Anfrage enthalten sein darf.

<include>
: Nur die angegebenen Attribute sollen in der Ergebnismenge enthalten sein.

<exclude>
: Die angegebenen Attribute dürfen in der Ergebnismenge nicht enthalten sein.

6.9.3 Weitere Angaben

Neben der Objektselektion (**<restriction>**) und dem Attributsfilter (**<filter>**) gibt es noch weitere Angaben, mit denen eine Anwendung eine Anfrage genauer spezifizieren kann.

<nearestNeighbor> *(optional)*
: Mit diesem Element kann die Ergebnismenge auf naheliegende Objekte beschränkt werden. Angegeben wird ein numerischer Wert n, ein **<target>** (welches räumliche Attribut für die Entfernungsberechnung verwendet werden soll) und ein **<referenceValue>**, der angibt, von welchem Punkt aus die Entfernung berechnet werden soll. Die Objekte werden nach dieser Entfernung sortiert und die ersten n Objekte werden zurückgegeben.

<aa> *(optional)*
: Mit diesem Element können lokale Umgebungsmodelle (***a****ugmented* ***a****reas)* angegeben werden, in denen die Ergebnisobjekte liegen müssen. Dadurch wird gleichzeitig die Datenquelle ausgewählt, da jedes lokale

Umgebungsmodell nur auf einem Server zu finden ist. Werden ein oder mehrere **<aa>**-Elemente angegeben, so findet keine Datenquellenauswahl durch den räumlichen Verzeichnisdienst mehr statt. Stattdessen wird die Anfrage direkt an die angegeben Umgebungsmodellserver weitergeleitet (vgl. Schritt 3 in Kapitel 5.3.1).

<ecs> *(optional)*
In Kapitel 7.1 wird erläutert, wie das Umgebungsmodell durch sogenannte erweiterte Schemata erweitert werden kann. Mit dem **<ecs>**-Element (***e**xtended **c**lass **s**chema)* kann ein Schema angeben werden, das die Anwendung versteht. Damit können in der Ergebnismenge auch Objekte enthalten sein, die nicht im Standardschema des Umgebugsmodells definiert werden (vgl. Schritt 8 in Kapitel 5.3.1).

<srs>
Die Nexus-Plattform unterstützt verschiedene räumliche Koordinatensysteme. Eine Anwendung muss mit diesem Element eine Liste von Koordinatensystemen (***s**patial **r**eference **s**ystems)* angeben, die sie versteht

<geoDataFormat>
Räumliche Daten können im Umgebungsmodell entweder als GML oder als WKT-Daten dargestellt werden (siehe Kapitel 6.6.1). Mit diesem Element kann die Anwendung angeben, welche geographischen Datenformate sie versteht. Die Nexus-Plattform muss Daten eines anderen Formats dann gegebenenfalls umwandeln.

6.9.4 Daten-Manipulation

Neben der Selektion können mit AWQL auch Daten manipuliert, d.h. eingefügt, geändert oder gelöscht werden (siehe Kapitel 5.3.2, 5.3.3 und 5.3.4). Zum Einfügen neuer Objekte kann direkt AWML verwendet werden. Das Ergebnis einer Datenmanipulation ist eine Liste von Objektidentifikatoren (*NOLs*) und eine Angabe, ob die Manipulation erfolgreich war oder nicht. Dies wird in der *Change Report Language* angegeben, die in [BDG+04] spezifiziert wird.

Für die Datenmanipulation stehen folgende Elemente zur Verfügung, von denen höchstens eines in einem AWQL-Dokument auftreten darf:

<update>
Das Element enthält Attribute und deren neuen Werte, die nun bei den in der Objektselektion ausgewählten Objekten eingetragen werden.

<append>
Das Element enthält Attribute und deren neuen Werte. Anstatt die alten Attribute zu überschreiben, wird ein weiteres Attribut mit gleichem Namen dem Objekt hinzugefügt.

<delete>
Das Element gibt an, dass die bei der Objektselektion ausgewählten Objekte gelöscht werden sollen.

6.9.5 Einschränkungen für globale Anfragen

Mit AWQL können Anfragen mit oder ohne räumliche Einschränkung formuliert werden. Richtet sich die Anfrage an einen einzelnen Umgebungsmodellserver (*lokale Anfrage*), so sind alle diese Anfragen auch zulässig. Die Föderation (Kapitel 5.2.3) benötigt jedoch Informationen, nach denen sie die Umgebungsmodellserver auswählen kann, die eine Anfrage beantworten können. Deswegen gilt für globale Anfragen die Einschränkung, dass eine AWQL-Anfrage mindestens eine der folgenden Angaben enthalten muss:

- für alle Ergebnisobjekte: entweder eine räumliche Restriktion auf die Ergebnisobjekte (**<within>** oder **<overlaps>**) oder eine Anfrage auf die Objekt-ID der Ergebnisobjekte (**NOL**),
- eine Beschränkung der Anfrage auf ein oder mehrere lokale Umgebungsmodelle (**<aa>** - Angabe),
- eine Beschränkung auf eine Anzahl nächstgelegener Objekte (**<nearestNeighbor>**).

6.9.6 AWQL-Beispiel

In Abbildung 25 ist ein Beispiel für eine AWQL-Anfrage zu sehen. Es handelt sich um die Lösung für das *famous italian restaurant problem*, das in zahlreichen Publikationen zu finden ist: „Wo sind die nächsten italienischen Restaurants?“ Werte, die dabei *kursiv* gedruckt sind, sind dabei Platzhalter für die Originaldaten, die für diese Darstellung hier zu umfangreich gewesen wären.

Das **<awml>-**Element (Zeilen 1 und 39) ist das Wurzel-Element der Anfrage, in dem auch zusätzliche Namensräume für erweiterte Schemata definiert werden können (siehe dazu auch Kapitel 7.1); in diesem Fall handelt es sich um den Namensraum *sr*, der in Zeile 20 verwendet wird, um einen Typ anzugeben, der im Standard-Schema des Umgebungsmodells nicht vorkommt. Die Zeilen 2 und 3 geben zwei lokale Umgebungsmodelle an, aus denen die Anfrage beantwortet werden soll. Für diese Anfrage findet also keine dynamische Serverauswahl statt.

In Zeile 4 wird angegeben, dass in der Ergebnismenge Objekte aus einem erweiterten Schema enthalten sein dürfen (dessen Namensraum im Wurzelelement definiert wurde). Zeile 5 gibt an, dass die räumlichen Daten im WKT-Format zurückgegeben werden sollen.

```
1:  <awql xmlns:sr="nexus://nexusschemas.org/SpecialtyRestaurants">
2:    <aa>nexus://guide.stuttgart.de/gourmet/...</aa>
3:    <aa>nexus://michelin.com/europe/...</aa>
4:    <ecs> nexus://nexusschemas.org/SpecialtyRestaurants </ecs>
5:    <geoDataFormat> WKT </geoDataFormat>
6:    <restriction>
7:      <or>
8:        <and>
9:          <equal>
10:           <target> type.value </target>
11:           <referenceValue> restaurant </referenceValue>
12:         </equal>
13:         <equal>
14:           <target> style.value </target>
15:           <referenceValue> italian </referenceValue>
16:         </equal>
17:       </and>
18:       <equal>
19:         <target> type.value </target>
20:         <referenceValue> sr:ItalianRestaurant </referenceValue>
21:       </equal>
22:     </or>
23:   </restriction>
24:   <nearestNeighbors num="5">
25:     <target> pos.value </target>
26:     <referencePoint>
27:       <nsat:WKT>...WKT representation of my position value... </nsat:WKT>
28:     </referencePoint>
29:   </nearestNeighbors>
30:   <include>
31:     <target> pos.value </target>
32:   </include>
33:   <include>
34:     <target> menu.value </target>
35:   </include>
36:   <include>
37:     <target> name.value </target>
38:   </include>
39: </awql>
```

Abbildung 25: AWQL-Beispiel (nach [BDG+04])

Soviel zu den allgemeinen Angaben. Die Zeilen 6-23 spezifizieren die Objektselektion. Gesucht werden alle Objekte, die entweder vom Typ *restaurant* sind und deren Stil *italian* ist, oder die vom Typ *ItalianRestaurant* sind, der im erweiterten Schema definiert wurde.

Diese Objekte werden durch die Angaben in den Zeilen 24-29 der Entfernung ihrer Position zu einer gegebenen Position (welche die Anwendung beispielsweise von einem GPS-Sensor erhalten hat) sortiert und davon die ersten fünf ausgewählt.

Die folgenden drei **<include>**-Elemente (Zeilen 30-38) dienen schließlich dazu, von diesen fünf Objekten nur die Position, die Speisekarte und den Namen zurückzugeben.

6.10 Zusammenfassung

In diesem Kapitel wurde das Umgebungsmodell mit seinen grundlegenden Eigenschaften, seinen Attributs- und Objekttypen und seinen Serialisierungs- und Anfragesprachen vorgestellt. Um zu zeigen, dass dieser Entwurf auch der Anforderung 12 (Erweiterbarkeit des Umgebungsmodells) genügt, werden nun im nächsten Kapitel Erweiterungsmöglichkeiten und -Arbeiten beschrieben.

Nur wer sorglos in die Zukunft blicken kann, genießt mit gutem Gefühl die Gegenwart. (Stefan Zweig)

Kapitel 7: Erweiterungen des Umgebungsmodells

In diesem Kapitel geht es darum, die Anforderung 12 (Erweiterbarkeit des Umgebungsmodells) zu erfüllen. Dazu wird zunächst in Kapitel 7.1 dargestellt, wie das Umgebungsmodell selbst erweitert werden kann. In Kapitel 7.2 werden dann drei Arbeiten vorgestellt, in denen die Plattform und auch das Modell durch die Integration bestehender Systeme erweitert wurde (siehe auch Anforderung 4 (Integration bestehender Informationsdienste)). Kapitel 7.3 gibt schließlich einen Ausblick auf zukünftig geplante Weiterentwicklungen des Modells.

7.1 Erweiterte Schemata

Wie schon in Kapitel 6.2.12 angedeutet, kann das Schema des Umgebungsmodells durch sogenannte *erweiterte Schemata* ergänzt werden. Dies ist insbesondere dann sinnvoll, wenn eine Datenquelle Daten anbieten möchte, die nicht im Umgebungsmodell abbildbar sind, und wenn es auch eine (geplante) Anwendung gibt, die diese Daten vermutlich anfragen wird.

Eine Möglichkeit wäre es, die zusätzlichen Daten einfach über eine gesonderte Schnittstelle oder in einem proprietären Schema anzubieten. In diesem Fall könnte jedoch nicht mehr von den Vorteilen der Nexus-Plattform profitiert werden: für andere Anwendungen wären diese Daten vollkommen unsichtbar, und die neue Anwendung müsste die Integration mit bestehenden Daten des Umgebungsmodells selbst vornehmen. Deswegen bietet es sich an, das Datenschema der zusätzlichen Daten zu veröffentlichen und Vererbungs-Beziehungen zum Standardschema zu definieren.

Dazu werden im erweiterten Schema neue Objekttypen definiert, die von mindestens einem Objekttyp des Standard-Schemas erben müssen (siehe Abbildung 9 auf Seite 115 sowie die formale Definition in Kapitel 6.4.4). Damit können neue Anwendungen, sofern sie von dem erweiterten Schema Kenntnis haben, auf diese zusätzlichen Daten zugreifen. Doch auch bestehende Anwendungen profitieren von einem erweiterten Schema: da eine Vererbungsbeziehung zu bekannten Objekttypen angegeben ist, kann die Nexus-Plattform bei einer Anfrage auch die Objekte eines erweiterten Schemas zurückgeben. Da die Anwendung jedoch nicht dafür entwickelt wurde, die Daten des erweiterten Schemas zu verarbeiten, werden solche Objekte transformiert, und zwar in eine Instanz des speziellsten Typs des Standard-Schemas. Zusätzliche Attribute werden dabei weggelassen.

Wie in Kapitel 6.5 erläutert wurde, gibt es verschiedene Schemadateien, mit denen das Umgebungsmodell definiert wird:

- Das Klassenschema *Nexus Standard Class Schema (NSCS)* definiert die Namen der Objekttypen und ordnet ihnen Attribute zu.
- Das Attributsschema *Nexus Standard Attribute Schema (NSAS)* definiert einfache und komplexe Attributtypen, die auf die Basisdatentypen abgebildet werden und ordnet Attributsnamen Attributtypen zu.
- Das Basistypenschema *Nexus Standard Attribute Types (NSAT)* definiert die Basisdatentypen, die einfache Attributtypen oder die Blätter in komplexen Attributtypen haben können.

Um ein erweitertes Schema anzulegen, sind folgende Schritte notwendig:

- Für die neuen Objekttypen wird ein erweitertes Klassenschema angelegt. Dieses importiert das *NSCS* und definiert dann die neuen Objekttypen analog zu den bestehenden Objekttypendefinitionen im *NSCS*. Jeder neue Objekttyp muss direkt oder transitiv von einem Objekttyp im *NSCS* erben. Die Wahl eines geeigneten Vaterobjekttyps obliegt dem Schemamodellierer: er sollte sich dabei von der Semantik der Vererbung zwischen Objekttypen leiten lassen (siehe Kapitel 6.2.4). Denn wenn eine Anwendung, die sein erweitertes Schema nicht kennt, eine Anfrage nach Objekten des Vatertyps fragt, werden die Kindstypen in diesen Typ umgewandelt (durch Weglassen unpassender Attribute, siehe Kapitel 5.3.1). Deswegen sollten die erbenden Objekttypen auch semantisch eine „ist-ein-Beziehung“ zum Vatertyp haben.

- Werden für die neuen Objektstypen zusätzliche Attribute benötigt, so muss zusätzlich ein erweitertes Attributsschema angelegt werden, das vom erweiterten Klassenschema importiert wird und selbst das *NSAS* importiert. Dort können nun neue Attributsnamen und einfache oder komplexe Attributtypen angelegt werden, die dann von den neuen Objektstypen im erweiterten Klassenschema verwendet werden.
- Sollten die vom *NSAT* definierten Basistypen nicht ausreichen, so muss zusätzlich noch ein erweitertes Basistypenschema angelegt werden, in dem diese definiert werden. Dies hat jedoch in der Regel Auswirkungen auf die Implementierung von Komponenten, die das erweiterte Schema benützen sollen. Es ist zwar möglich, die Verarbeitung von Umgebungsmodelldaten so zu implementieren, dass sie mit beliebigen (aber bekannten) Klassen- und Attributsschemata zurecht kommt. Dazu wird das Attributsschema geparst und die Verarbeitung der Attribute und Attributsteile (bei komplexen Attributtypen) jeweils auf eine feste Menge von Operatoren und Funktionen zurückgeführt, welche die Basistypen verstehen und implementieren. Sobald nun neue Basistypen dazu kommen, muss diese Verarbeitung erweitert werden. Dies gilt mindestens für den Umgebungsmodellserver, auf dem die erweiterten Daten gespeichert sind, und für die Anwendung, die mit ihnen umgehen soll. Sollen jedoch Mehrwertdienste, andere Umgebungsmodellserver oder die Föderation solche Daten verwalten, muss auch dort die Implementierung erweitert werden, was im Allgemeinen sehr aufwändig oder nicht möglich ist (sofern der Anbieter der neuen Daten keinen Zugriff auf die Komponenten hat).

7.2 Integrationsstudien

In diesem Kapitel werden mehrere Arbeiten vorgestellt, in denen bestehende Informationssysteme in die Nexus-Plattform integriert wurden. Dabei wird besonders darauf eingegangen, wie die Erweiterbarkeit des Nexus-Ansatzes von Nutzen war.

7.2.1 Aware Home Spatial Server (AHSS)

Im Projekt *The Aware Home* [KOA+99] der Georgia Tech werden Möglichkeiten zur (kontextbezogenen) Computerunterstützung im eigenen Heim erforscht. Fernziel ist beispielsweise die Unterstützung von (allein lebenden) Senioren; das System soll sowohl die Lebensqualität verbessern als auch Krisensituationen erkennen und entsprechende Notdienste verstän-

digen. An der Georgia Tech wurde ein Zwei-Familien-Haus als *living laboratory* für das Projekt gebaut, in dem zahlreiche Sensor- und Aktorsysteme sowie eine technische Infrastruktur für die Entwicklung kontextbezogener Anwendungen zur Verfügung stehen.

Bisher wurden die verschiedenen Anwendungen von den Entwicklern mit den entsprechenden Sensorsystemen „hart verdrahtet", d.h. eine Abstraktion im Sinne eines Umgebungsmodells oder einer gemeinsamen Datenverwaltung existierte nicht. Deswegen wurde dort der *Aware Home Spatial Server* (*AHSS*) entwickelt, eine lokale Datenbank-Lösung, die ein sehr einfaches Kontextmodell bereitstellt. Die Semantik der Informationen ist nicht expliziert festgelegt, sondern steckt in den Anwendungen bzw. wird durch Absprachen zwischen den Entwicklern definiert.

Als Integrationsstudie sollte nun der *AHSS* als räumlicher Umgebungsmodellserver in die Nexus-Plattform integriert werden. Dazu wurde ein Wrapper entwickelt, der zum einen die notwendigen Schnittstellen für Nexus-Anwendungen zur Verfügung stellt, zum anderen auch eine semantische Abbildung vom einfachen Modell des *AHSS* in das Umgebungsmodell vornimmt. Damit wurde die Erweiterbarkeit des Nexus-Ansatzes für bestehende Datenquellen, die bereits ortsbezogene Daten verwalten, gezeigt. Im Folgenden werden nun kurz die wesentlichen Punkte dieser Arbeit diskutiert. Eine ausführlichere Darstellung findet sich in [LBB+04] bzw. [Leh03].

Integrations-Szenario

Bei der Integration der *AHSS*-Lösung in Nexus wurde von folgendem Szenario ausgegangen: Ein Wohnhaus wurde durch Einbau verschiedener Sensoren, Aktoren und Infrastruktur zu einer instrumentierten Umgebung. Die Besitzerin des Hauses – eine Informatikerin – betreibt einen *AHSS*, der verschiedene Anwendungen in ihrem Haus mit Daten versorgt, z.B. ein sprachgesteuertes Kommunikationssystem, das je nach Situation und Ort des Angerufenen geeignete Medien für die Verbindung wählt, eine Türklingel, die ein Videobild des Besuchers auf ein nahes Präsentationsmedium überträgt und bei Einlass den Besucher für bestimmte Räume im Gebäude autorisiert und ein Alarmsystem, das nicht-autorisierte Personen im Gebäude feststellt und meldet. Sie hört von Nexus, einer offenen, globalen Plattform für kontextbezogene Anwendungen, und möchte gerne daran teilhaben: zum einen möchte sie eine Nexus-

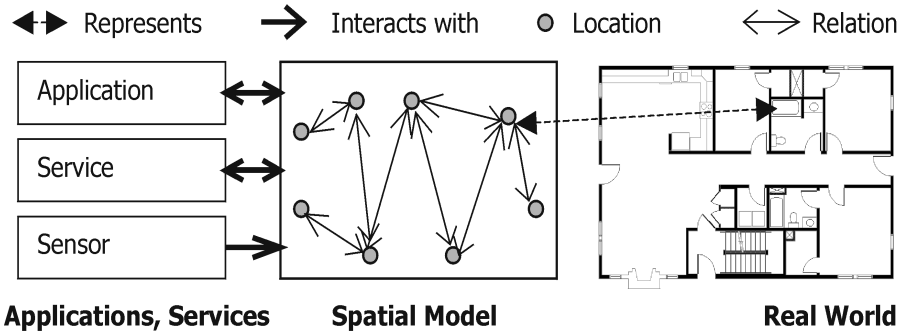

Abbildung 26: Konzept des AHSS (aus [LBB+04])

Anwendung kaufen, die bei einem Brand den Wohnungsplan und die Position der Menschen im Haus an die Feuerwehr übermittelt, zum anderen möchte sie ihren Freunden ermöglichen, dass diese ihre persönlichen Nexus-Anwendungen auch in ihrem Haus verwenden können. Allerdings möchte sie keinen Umgebungsmodellserver installieren: das Umgebungsmodell ihres Hauses wird ja bereits vom *AHSS* verwaltet, und ihre bestehenden Anwendungen sollen weiterhin damit arbeiten („*never change a running system*“). Wie kann sie nun die Daten des *AHSS* für Nexus-Anwendungen und die Föderation zur Verfügung stellen?

Aufbau des *AHSS*

Abbildung 26 zeigt die Konzeption des AHSS. Das räumliche Modell (*Spatial Model,* Mitte) besteht aus allgemeinen *Locations*, die ortsbezogenen Objekten entsprechen, z.B. einem Raum, einer Kamera oder einer Person. Sie sind über Beziehungen (*Relation*) miteinander verbunden. *Locations* haben eine Identität, einen Namen, ein räumliches Attribut und darüber hinaus beliebige Attribut-Wert-Paare, mit denen Daten über die Objekte gespeichert werden. Sensoren aktualisieren diese Daten und Dienste und Anwendungen können über verschiedene Schnittstellen darauf zugreifen: implementiert sind eine *Corba*-Schnittstelle, RMI-IIOP und eine spezielle *AHSS* Bibliothek, mit der Java-Anwendungen mit dem *AHSS* komfortabel interagieren können. Der *AHSS* speichert die Daten in einer räumlichen Datenbank und setzt die Anfragen der Anwendungen in SQL um.

AHSS als Umgebungsmodellserver?

Bei der Verwendung des *AHSS* als Umgebungsmodellserver ergeben sich folgende Probleme:

- Schnittstelle: Der *AHSS* muss eine AWQL/AWML-Schnittstelle über SOAP bereitstellen, um von der Föderation und von Nexus-Anwendungen abgefragt werden zu können. Die Schnittstellen des *AHSS* sollen jedoch für bestehende Anwendungen erhalten bleiben.
- Datenmodell: Das Schema des *AHSS* ist sehr flexibel. Definiert sind nur *Locations*, die beliebige Attribute haben können. Damit kann zwar das Umgebungsmodell auf das *AHSS*-Modell abgebildet werden, andersherum ist das jedoch im Allgemeinen nicht möglich.
- Konformität: Die AWQL-Schnittstelle sieht vor, dass nur die Objekte zurückgegeben werden, die dem Umgebungsmodell bzw. den in der Anfrage angegebenen erweiterten Schemata entsprechen. Da bestehende *AHSS*-Anwendungen jedoch weiterhin über ihre Schnittstellen auf den Server zugreifen, können sich dort beliebige Objekte befinden oder nicht-konforme Attribute zu bestehenden Objekten hinzugefügt werden.
- Vollständigkeit: Es soll möglich sein, auf sämtliche Objekte des *AHSS* auch über die Nexus-Schnittstelle zugreifen zu können. Es könnte sein, dass bestehende *AHSS*-Anwendungen auf die AWQL-Schnittstelle umgestellt werden, um von den Vorteilen der Föderation zu profitieren und auch in anderen Häusern zu funktionieren. Trotzdem möchten sie weiterhin ihre speziellen *AHSS*-Objekte verwenden können.

Integration in Nexus

Für die Lösung der oben genannten Probleme wurde ein Wrapper entwickelt, der die AWQL/AWML-Schnittstelle implementiert.

- Schnittstelle: Der Wrapper bietet eine AWQL-Schnittstelle an und beantwortet die Anfragen in AWML. Dazu wird das AWQL geparst und intern auf räumliches SQL umgesetzt, mit dem die verwendete Datenbank abgefragt wird.
- Datenmodell: Objekte, die über die AWQL-Schnittstelle in den *AHSS* eingefügt werden, werden als dort als *Locations* gespeichert. Die räumlichen Attribute können als räumliche Datentypen umgesetzt werden, die **NOL** auf die Objekt-ID. Alle anderen werden in einer *String*-Repräsentation gespeichert, die über die XML-Serialisierung zur Verfügung steht. Über das **type-**Attribut können später auf diese Weise eingefügte Objekte wieder auf die entsprechenden Umgebungsmodell-Objekttypen abgebildet werden. Bestehende *Locations*, die im Allgemeinen kein

type-Attribut haben, werden auf den Objekttyp **SpatialObject** abgebildet, das räumliche Attribut auf **pos** bzw. **extent** (je nachdem, ob es sich um einen Punkt oder eine Ausdehnung handelt). Wenn keine Position enthalten ist, kann diese aus dem **extent** generiert werden (z.B. der Mittelpunkt).

- Konformität: Der Wrapper kann Umgebungsmodell-Schemata einlesen und kann damit entscheiden, welche Objekttypen und welche Attribute für eine Anfrage zulässig sind. Attribute, die nicht über die Nexus-Schnittstelle eingefügt wurden (und damit i.A. auch nicht dem Umgebungsmodell entsprechen) müssen dann aus dem AWML-Ergebnis gefiltert werden. Dies trifft für alle Attribute allgemeiner *Locations* zu, die über die ID, den Namen und das räumliche Attribut hinausgehen.
- Vollständigkeit: Durch die Einschränkungen der Konformität sind viele Daten nicht zugreifbar. Dafür wurde das Konzept der *generischen erweiterten Schemata* entwickelt: der *AHSS* bietet neben dem Standardschema und beliebigen installierten erweiterten Schemata auch ein spezielles ECS an. Dieses definiert nur einen einzigen Objekttyp (**AHSSLocation**), der direkt von **SpatialObject** erbt. Das besondere an diesem Objekttyp ist, dass er beliebige Attribut-Wert-Paare enthalten kann (XML-Typ **any**). Damit können alle *Locations*, die im *AHSS* gespeichert werden, auch mit AWQL abgefragt und mit AWML zurückgegeben werden.

Bewertung

Über seine bestehenden Schnittstellen konnte der *AHSS* die Anforderungen typischer *AwareHome*-Anwendungen voll erfüllen (auf einem handelsüblichen Laptop bei 1000 verwalteten *Locations* 58.2 Anfragen pro Sekunde bei einer Anwendung, bei sechs Anwendungen immerhin noch 11 Anfragen pro Sekunde pro Anwendung). Über den Wrapper war natürlich mit einer Einschränkung der Performanz zu rechnen. Der Durchsatz sank auf etwa ein Sechstel (10 Anfragen pro Sekunde), was für viele Anwendungsfälle immer noch ausreichend ist. Mit einer effizienteren XML-Verarbeitung sollte dies noch deutlich verbessert werden können. Außerdem können Anwendungen, die ständig im Haus laufen, über die bestehenden, effizienteren Schnittstellen auf den *AHSS* zugreifen

(und trotzdem das Umgebungsmodell verwenden). Die Wrapper-Schnittstelle würde dann nur für die (selteneren) Anfragen benötigt, die von externen Anwendungen stammen.

Mit der Arbeit konnte gezeigt werden, dass das Umgebungsmodell und seine Verwaltung flexibel genug sind, um andere Datenquellen und Modelle einzubinden. Auf der „grünen Wiese“ funktionierte die Schemaumsetzung sehr gut. Im praktischen Einsatz des vorgestellten Szenarios ist jedoch davon auszugehen, dass es viele *Locations* gibt, die von mehreren bestehenden Anwendungen geteilt werden und deshalb in ihrem Aufbau recht stabil sind. Für diese würde es sich anbieten, den Wrapper um eine konfigurierbare Schema-Mapping-Komponente zu erweitern, mit der Entwickler ihre bestehenden *Locations* auf speziellere Objekttypen des Umgebungsmodells abbilden können (z.B. **Person** oder **Room**).

7.2.2 Lokationsdienst

Der Lokationsdienst entstand in der Dissertation von Alexander Leonhardi [Leo03] als erste Komponente im Nexus-Projekt. Er ist in der Lage, viele mobile Objekte effizient zu verwalten. Zu diesem Zeitpunkt gab es noch kein übergreifendes Plattformkonzept, d.h. kein umfassendes Umgebungsmodell und auch keine Spezifikation der AWQL/AWML-Schnittstelle. Deswegen bietet er eine eigene, funktionale Schnittstelle. Erste Nexusanwendungen (z.B. der *NexusScout* [NGS03]) verwendeten später diese proprietäre Schnittstelle, um mobile Objekte zu verwalten, neben der AWQL-Schnittstelle für den Zugriff auf stationäre Daten.

Um einen einheitlichen Zugriff auf mobile und stationäre Daten über die AWQL-Schnittstelle zu ermöglichen, wurde ein Wrapper [Mot03] entwickelt, der AWQL auf die funktionale Schnittstelle des Lokationsdienstes umsetzt.

Aufbau des Lokationsdiensts

Der Lokationsdienst wurde speziell auf die Eigenschaften mobiler Objekte zugeschnitten [GBH+05]. Diese werden mit einer ID, einen Namen, eine Position und beliebige weitere Attribut-Wert-Paaren (ähnlich den *Locations* des *AHSS*, siehe Kapitel 7.2.1*)* verwaltet.

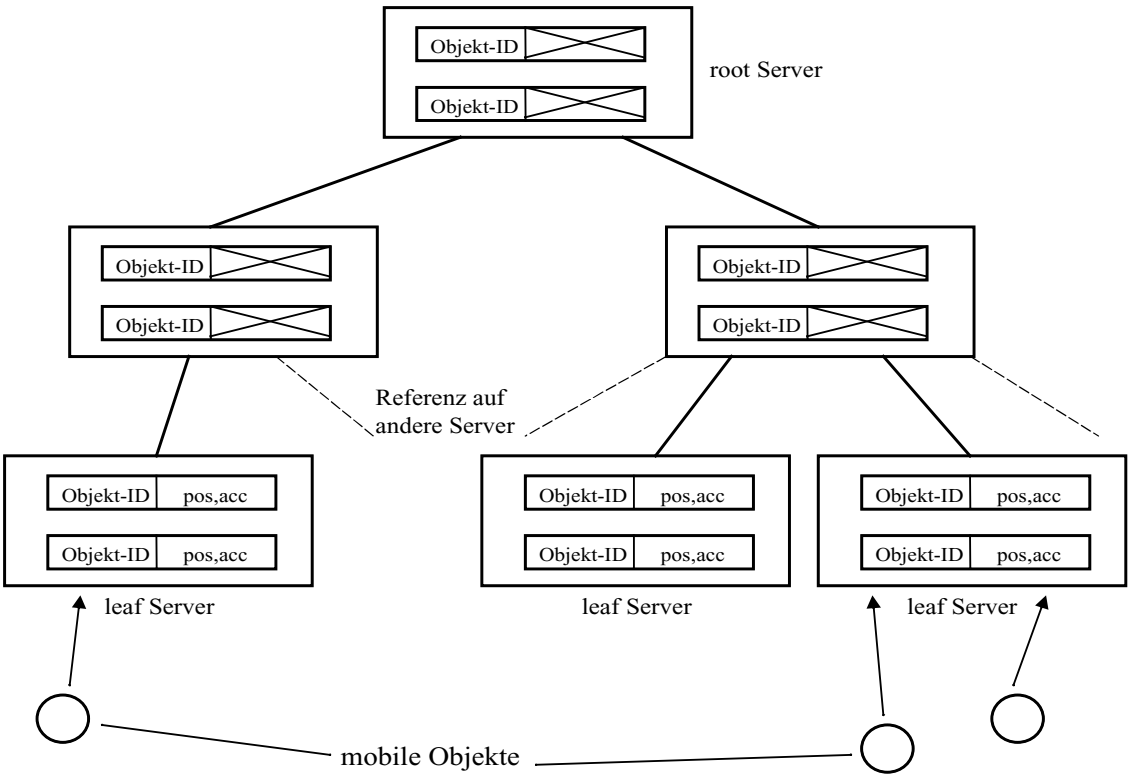

Abbildung 27: Aufbau des Lokationsdienst (nach [Leo03])

Da mobile Objekte ihre Position häufig aktualisieren, muss die Verwaltung bei einer großen Anzahl von Objekten auf mehrere Server verteilt werden. Unter der Lokalitätsannahme (siehe Kapitel 2.2) bietet sich eine räumliche Verteilung an.

Abbildung 27 zeigt den Aufbau des Lokationsdienst. Er besteht aus einer Hierarchie von einzelnen Lokationsservern, die unter sich den Raum aufteilen. Der Wurzelknoten (*root server*) ist für das gesamte Gebiet zuständig. Kindknoten teilen das Gebiet ihres Vaterknoten jeweils vollständig untereinander auf. Nur die Blattserver verwalten tatsächlich die mobilen Objekte: alle anderen enthalten lediglich Referenzen, welche Objekte von welchen ihrer Kindknoten verwaltet werden.

Die Aktualisierungen eines mobilen Objekts werden in der Regel an den Server geschickt, der es gerade verwaltet, in dessen Dienstgebiet es sich also befindet. Wenn es dieses verlässt, erhält der Server einen Wert, der außerhalb liegt. Dann leitet er die Aktualisierung an seinen Vaterknoten weiter, der sie entweder an einen benachbarten Kindsknoten weitergibt

oder sie weiter nach oben leitet, bis ein Server gefunden ist, in dessen Dienstgebiet die neue Position liegt. Das Objekt wird an diesen übergeben (*handover)* und die Anwendung, von der die Aktualisierung kam, informiert, dass nun ein neuer Server zuständig ist. Tabelle 5 gibt einen

Tabelle 5: Funktionen des Lokationsdiensts

Funktion	Beschreibung	AWQL/AWML-Schnittstelle
PositionQuery	liefert die Position eines gegebenen Objekts	AWQL-Anfrage nach dem Attribut **pos**
RangeQuery	liefert die Objekte innerhalb eines gegebenen Gebiets	AWQL: Prädikat **overlap** mit **pos**
NearestNeighbors	liefert das Objekt, das zu einer gegebenen Position am nächsten liegt	AWQL: **<nearest n=1>**
registerObject	fügt ein neues Objekt in den Lokationsdienst ein	AWQL: **<insert>**
deregisterObject	löscht ein Objekt aus dem Lokationsdienst	AWQL: **<delete>**
handover	übergibt ein Objekt an einen neuen Lokationsserver	**<insert>** bei neu, **<delete>** bei alt

Überblick über die funktionale Schnittstelle des Lokationsdienstes und seine grundsätzliche Abbildung auf die AWQL/AWML-Schnittstelle. Der Lokationsdienst unterstützt Positionen mit gegebener Ungenauigkeit und berücksichtigt dies auch in allen Anfragen, was an dieser Stelle vernachlässigt wird.

Lokationsdienst als Umgebungsmodellserver?

Bei der Verwendung des Lokationsdienstes als Umgebungsmodellserver ergeben sich zum Teil ähnliche Probleme wie beim *Aware Home Spatial Server (AHSS,* siehe Kapitel 7.2.1), zum Teil sind diese jedoch auch spezifisch für den Lokationsdienst:

- Systemmodell: Soll der gesamte Lokationsdienst als ein Umgebungsmodellserver auftreten oder die einzelnen Lokationsserver? Dies hat Auswirkungen auf die Wahl des Dienstgebiets und die Registrierung im räumlichen Verzeichnisdienst (Kapitel 5.2.2).

- Schnittstelle: Im Unterschied zum *AHSS* ist im Lokationsdienst keine allgemeine Anfrageschnittstelle vorgesehen, sondern nur Funktionen (siehe Tabelle 5). Diese können zwar einfach in AWQL umgesetzt werden, andererseits muss der Wrapper auch beliebige andere AWQL-Anfragen bearbeiten können.
- Datenmodell: Die Abbildung des Datenmodells ist analog zu der des *AHSS*, wobei die *Locations* dort den mobilen Objekten hier entsprechen.
- Wiederauffindbarkeit von Objekten: Der *Nexus Object Locator (NOL*, siehe Kapitel 6.2.2) enthält neben einer eindeutigen Objekt-ID auch einen Anteil, der den Server angibt, auf dem das zugehörige Objekt verwaltet wurde. Da sich mobile Objekte naturgemäß bewegen, wechseln sie den Server, was dazu führt, dass sie auch ihre NOL ändern würden.
- Konformität: Auch dieses Problem stellt sich im Lokationsdienst wie im *AHSS* gleichermaßen.
- Vollständigkeit: Der Zugriff auf bestehende, nicht modellkonforme Daten ist beim Lokationsdienst nicht so relevant, da es wenig bestehende Anwendungen gibt, die darüber hinaus bereits über AWQL-Schnittstellen verfügen (für ihre stationären Daten). Es wurde allerdings eine ähnliche Lösung wie beim *AHSS* umgesetzt, um den Migrationsaufwand möglichst gering zu halten.

Integration in Nexus

Auch der Lokationsdienst wurde mit einem AWQL-Wrapper in die Nexus-Plattform integriert. Dabei wurden folgende Entwurfsentscheidungen getroffen:

- Systemmodell: Der Wrapper ist für einzelne Lokationsserver zuständig, nicht für den gesamten Lokationsdienst. Dies ist deswegen sinnvoll, da ansonsten eine doppelte Serverauswahl getroffen wird: zunächst über den räumlichen Verzeichnisdienst und dann noch einmal über die Hierarchie des Lokationsdienstes. Allerdings kann ein Wrapper für mehrere Lokationsserver zuständig sein, falls diese sehr kleine Dienstgebiete haben. Der Wrapper führt eine Tabelle, welcher Lokationsserver gerade welches Objekt verwaltet (ähnlich einem Lokationsserver, der kein Blattknoten ist). Wird für das Objekt ein *Handover* durchgeführt, so aktualisiert der Wrapper diese Tabelle. Im Unterschied zum bisherigen Systemmodell des Lokationsdienst wird

dieses Wissen nicht mehr an die Anwendung weitergegeben, da dies der Lokationstransparenz der Nexus-Föderation widersprechen würde. Stattdessen kann eine Anwendung über den *Nexus Object Locator (*der ja auch einen Server-Anteil enthält) herausfinden, welcher AWQL-Wrapper für das Objekt zuständig ist.

- Schnittstelle: Der AWQL-Wrapper übersetzt AWQL auf die Funktionen des Lokationsservers, siehe Tabelle 5. Wie dort zu sehen ist, gibt es keine Funktion, um nach ortsunabhängigen Attributen (z.B. dem Typ) zu selektieren. Ohne Änderung des Lokationsservers müsste der Wrapper hierfür alle in Frage kommenden Objekte über die eine **RangeQuery** anfragen und dann selbst nach den weiteren Attributen filtern. Aus Effizienzgründen wurde an dieser Stelle der Lokationsserver um eine **Select-**Methode erweitert, welche die Filterung nach beliebigen Attributen erlaubt.
- Datenmodell: Das Datenmodell des Lokationsservers wurde analog zum *AHSS* auf das Umgebungsmodell umgesetzt. Allerdings werden nur **MobileObject** und dessen Kinder unterstützt.
- Wiederauffindbarkeit von Objekten: Die Objekt-ID eines mobilen Objekts wird auf die NOL abgebildet, die Server-ID stammt vom AWQL-Wrapper. Sollte das Objekt nun sich weiterbewegen und an einen anderen Lokationsserver weitergegeben werden, aktualisiert der Wrapper seine Tabelle – er weiß also immer noch, wo sich das Objekt befindet, auch wenn dieses außerhalb seines Dienstgebietes liegt. Wenn das Objekt nun über einen anderen AWQL-Wrapper abgefragt wird, erhält es eine andere NOL. Die Objekt-ID bleibt jedoch die selbe. Im ungünstigsten Fall erhält die Föderation zwei Repräsentationen des mobilen Objekts (vom alten AWQL-Wrapper und vom neuen), die jedoch über die gleiche Objekt-ID erkannt und zusammengeführt werden können. Für Anwendungen bietet es sich an, davon auszugehen, dass sich die NOL eines mobilen Objekts ändern kann und von Zeit zu Zeit eine neue Anfrage danach zu stellen.
- Konformität und Vollständigkeit: Diese Probleme wurden beim Lokationsdienst analog zum *AHSS* gelöst: auch dieser AWQL Wrapper kennt das Standard- und eventuelle erweiterte Schemata und bietet ein generisches erweitertes Schema für nicht konforme Attribute anderer Anwendungen an.

Bewertung

Die Performanz-Messungen für den AWQL-Wrapper wurden zweistufig ausgeführt, da der Wrapper nicht nur zusätzliche Funktionalität, sondern auch ein aufwändigeres Kommunikationsprotokoll (SOAP statt UDP) verwendet. Es ergab sich, dass die Anfragen über die SOAP-Schnittstelle 10mal so lange dauerten wie über UDP, und dass bei der Umsetzung von AWQL im Wrapper noch einmal ein Faktor drei hinzukam. Für die Anfrage einzelner Objekte mag diese Laufzeit zu akzeptieren sein, für viele Objekte ist dies jedoch zu langsam. Deswegen wird der Lokationsserver nun neu implementiert, mit einer eigenen AWQL-Schnittstelle, die dann auch durch entsprechende Indizes unterstützt wird. Es konnte mit dieser Arbeit jedoch gezeigt werden, dass auch der Lokationsdienst prinzipiell mit geringem Aufwand in das Umgebungsmodell und seine Verwaltung integriert werden kann.

7.2.3 KontextWiki

Ein Wiki ist ein System, bei dem Web-Seiten direkt über die Web-Schnittstelle editiert werden können. Hauptfokus ist dabei die einfache Erstellung und Manipulation von Inhalten. Die technologischen und organisatorischen Schranken werden möglichst gering gehalten. So ist es z.B. üblich, dass jeder Benutzer alles editieren kann. Die Inhalte werden nur durch die Gemeinschaft der Benutzer kontrolliert (siehe z.B. die freie Enzyklopädie [Wikipedia]).

Im Rahmen einer Diplomarbeit [Sch04] wurde untersucht, ob und wie sich die Wiki-Idee für ortsbezogene Anwendungen eignet. Das Wiki-System könnte dabei an zwei Stellen in der Nexus-Infrastruktur eingesetzt werden: als Nexus-Anwendung, die es mobilen und stationären Benutzern ermöglicht, über die Wiki-Schnittstelle Orte zu kommentieren und Umgebungsinformation abzulegen, und als Umgebungsmodellserver, der derart eingegebene Daten über die Nexus-Plattform anderen Anwendungen zur Verfügung stellt.

Architektur eines Wiki

Abbildung 28a) zeigt die grundlegende Architektur eines Wiki: der Benutzer greift über einen Webbrowser auf Wikiseiten zu, die von der Wiki-Software über einen Webserver im HTML-Format ausgeliefert werden. Die Wikiseiten werden in einem internen Format lokal gespeichert.

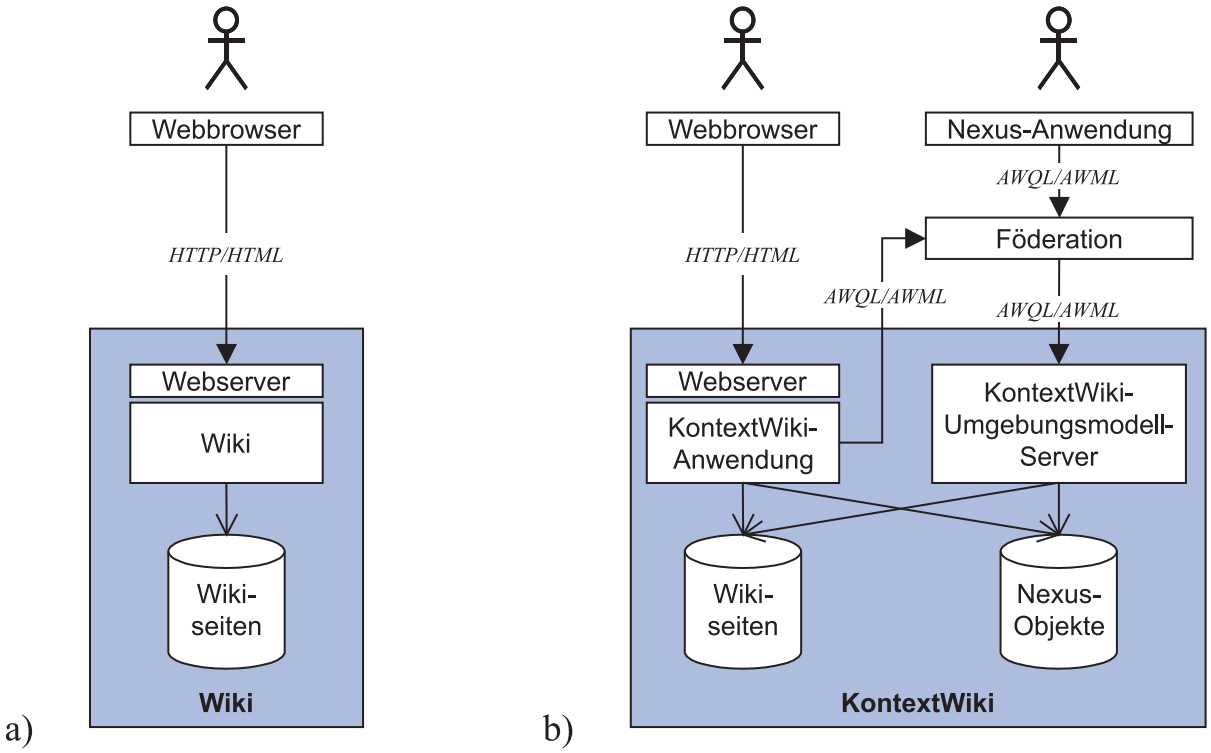

Abbildung 28: Wiki und KontextWiki Architektur

Wikiseiten haben einen eindeutigen Namen und könnten durch Wiki-URL plus Seitenname identifiziert werden. Ruft ein Web-Browser eine Wikiseite ab, wird sie aus dem internen Format nach HTML übersetzt und übertragen. Wenn unter dem angegebenen Namen keine Wikiseite existiert, so wird eine HTML-Seite generiert, mit welcher der Benutzer eine neue Wiki-Seite unter diesem Namen anlegen kann.

Der Inhalt einer Wikiseite besteht aus Text, der durch einfache Formatierungsanweisungen strukturiert werden kann. Einige Wiki-Implementierungen erlauben zusätzlich das Einbinden von Bildern.

Für die Entwicklung des KontextWiki wurde die Wiki-Implementierung *MoinMoin* [MoinMoin] in der Version 1.2 verwendet. Diese bietet ein *Plugin*-Konzept, das es ermöglicht, die Implementierung um neue Funktionen zu erweitern, ohne den eigentlichen Quellcode ändern zu müssen.

Folgende Komponenten stehen dabei für die Erweiterung zur Verfügung:

- *Parser* verarbeiten Textformate und übersetzen sie in die interne Darstellung (z.B. das Wiki-Textformat oder LaTeX).
- *Formatter* erzeugen aus der internen Darstellung ein bestimmtes Ausgabeformat (z.B. HTML oder XML).
- *Makros* können in normalen Wiki-Text eingebettet werden. Bei der Generierung von HTML werden sie durch dynamisch erzeugten Inhalt ersetzt (z.B. das aktuelle Datum oder ein Inhaltsverzeichnis).
- *Prozessoren* verarbeiten ganze Textblöcke im normalen Wiki-Text (z.B. eingeschobenen Quellcode).
- *Aktionen* betreffen die Verarbeitung ganzer Seiten (z.B. **SavePage**, das eine bearbeitete Seite speichert).

Abbildung 28b) zeigt die Erweiterung des Wiki zu einem KontextWiki. Dazu wird einerseits die Wiki-Software zu einer KontextWiki-Anwendung ausgebaut, mit der Benutzer im Wiki-Stil Umgebungsmodelldaten eingeben und editieren können. Zum anderen ermöglicht ein KontextWiki Umgebungsmodellserver den Zugriff auf diese Daten über die Föderation.

KontextWiki als Nexus-Anwendung

Die KontextWiki-Anwendung ist eine Erweiterung des Wiki. Auf Seite des Benutzers ändert sich nichts: er greift weiterhin mit einem Webbrowser über den Webserver auf das KontextWiki zu.

Um im KontextWiki ortsbezogene Daten eingeben zu können, wurden Makros und Prozessoren entwickelt, mit denen Objekte des Umgebungsmodells (im Folgenden: Nexus-Objekte) in eine Wiki-Seite eingebunden werden können. Die Nexus-Objekte des KontextWiki bilden ein lokales Umgebungsmodell. Sie werden beim Speichern der Seite in einer eigenen Speicherstruktur abgelegt (im Bild: „Nexus-Objekte“), damit für eine Anfrage auf die Nexus-Objekte des Wiki nicht alle Wikiseiten durchsucht werden müssen. Für diese Art der Speicherung wurde die **SavePage**-Aktion erweitert.

Dem Benutzer werden nun zusätzlich zu den übrigen Wiki-Funktionen folgende Möglichkeiten geboten:

- Eingabe von Daten: Es können Nexus-Objekte über Attribut-Wert-Paare eingegeben werden. Diese werden in einer eigenen Datenbank

abgelegt. Existiert zu einem Objekt noch kein NOL, so wird dieser von der KontextWiki-Anwendung angelegt und auch in den Text der Wiki-Seite eingefügt (über ein Attribut-Wert-Paar *NOL*="...").
- Erweiterung von Nexus-Objekten: Wenn dem Benutzer der NOL eines Objekts bekannt ist, so kann er auf beliebigen Wikiseiten weitere Information zu diesem Objekt anlegen. Diese werden dem Objekt in der Nexus-Objekte-Datenbank hinzugefügt.
- Einbetten von Anfragen: Ein Benutzer kann auf einer Wiki-Seite auch eine Anfrage an das Umgebungsmodell einbetten. Beim Generieren der HTML-Seite wird diese Anfrage beantwortet und das Ergebnis in die HTML-Darstellung eingebettet. Sollte die Anfrage auf die Nexus-Objekte des KontextWiki beschränkt sein (über eine **<aa>**-Angabe, siehe Kapitel 6.9.5), so wird sie aus der lokalen Nexus-Objekte-Datenbank beantwortet. Ansonsten wird sie als AWQL an die Föderation weitergeleitet und das Ergebnis verwendet. Die KontextWiki-Anwendung tritt hierbei als Nexus-Anwendung auf.

KontextWiki als Umgebungsmodellserver?

Die Nexus-Objekte des KontextWiki sollen auch der Föderation und damit anderen Nexus-Anwendungen zugänglich gemacht werden (Abbildung 28b)). Dafür greift ein KontextWiki-Umgebungsmodellserver auf die Nexus-Objekte-Datenbank der KontextWiki-Anwendung zu und liefert die Objekte auf AWQL-Anfragen an die Föderation. Bei dem Entwurf sind folgende Fragestellungen zu beachten:

- Systemmodell: Soll das gesamte KontextWiki als Umgebungsmodellserver auftreten oder nur einzelne Teile davon? Wie werden die Dienstgebiete der Teile oder auch des KontextWiki festgelegt?
- Schnittstelle: Bisher hat ein Wiki nur eine Benutzerschnittstelle, keine allgemeine Abfrageschnittstelle für Anwendungen. Diese muss also erst geschaffen werden.
- Datenmodell: Da die Daten in einem herkömmlichen Wiki unstrukturiert bzw. dokumentenorientiert sind, existiert kein Datenmodell. Im KontextWiki können Benutzer nun beliebige Objekte anlegen. Wie wird sicher gestellt, dass diese dem Umgebungsmodell entsprechen?
- Konformität: Da Benutzer grundsätzlich beliebige Datenobjekte eingeben können, stellt sich die Frage der Konformität analog zum AHSS und zum Lokationsdienst.

- Vollständigkeit: Da über die Wiki-Schnittstelle selbst alle Daten (ob strukturiert oder unstrukturiert) zugegriffen werden können und die Verwendung des KontextWiki als Umgebungsmodellserver nur einen Zusatz darstellt, kann auf die Vollständigkeit verzichtet werden.

Integration in Nexus

Für die Integration in Nexus wurden folgende Lösungen gewählt:

- Systemmodell: Es wurde zunächst davon ausgegangen, dass das gesamte KontextWiki einen Umgebungsmodellserver repräsentiert. Das Dienstgebiet kann auf zwei Arten festgelegt werden: entweder gibt es der Administrator des Wiki vor oder es wird in regelmäßigen Abständen aus den räumlichen Attributen der gespeicherten Nexus-Objekte berechnet und im räumlichen Verzeichnisdienst aktualisiert.
- Schnittstelle: In einer ersten, einfachen Lösung werden die Nexus-Objekte des KontextWiki im AWML-Format exportiert und können dann von einer bereits existierenden Umgebungsmodellserver-Implementierung eingelesen und über eine AWQL-Schnittstelle abgefragt werden. Grundsätzlich wäre jedoch auch eine Erweiterung der Wiki-Software um eine AWQL-Schnittstelle denkbar (so dass KontextWiki-Anwendung und der KontextWiki-Umgebungsmodellserver zusammenfallen).
- Datenmodell: Grundsätzlich können sämtliche Objekte des Umgebungsmodells über das KontextWiki eingegeben und, sofern dort vorhanden, auch abgefragt werden. Damit Benutzer wissen, welche Objekttypen existieren, gibt es eine menschenlesbare Version des *Standard Class Schema* als Wiki-Seite: dazu wird das XML-Schema mittels eines Transformationsskriptes in Wiki-Markup übersetzt und in eine Wiki-Seite eingebunden.
  Zudem kann jede Wiki-Seite selbst als **WebSituatedObject** repräsentiert werden. Dies ermöglicht einer Nexus-Anwendung nicht nur den Zugriff auf die Nexus-Objekte des KontextWiki, sondern auch auf die Wikiseiten selbst, die damit Teil des Umgebungsmodells werden.
- Konformität und Vollständigkeit: Für Schema-Erweiterungen, die von Benutzern gewünscht werden, ist eine Lösung ähnlich dem generischen Schema in AHSS oder Lokationsdienst denkbar: ein generisches KontextWiki-Schema erlaubt die Speicherung und Rückgabe beliebiger Objekte. Zudem könnte von Zeit zu Zeit ein erweitertes Schema aus den jeweiligen Objekten generiert werden, die nicht dem Stan-

dard-Schema entsprechen. Dies wurde jedoch bislang nicht implementiert.

Bewertung

Das KontextWiki stellt eine interessante Möglichkeit dar, Benutzern einfachen Zugriff auf Umgebungsmodelldaten zu ermöglichen und diese zu annotieren: sie benötigen dazu lediglich einen Webbrowser und müssen sich in die (sehr einfach gehaltene) Wiki-Syntax einarbeiten. Durch die einfache Lösung des „externen“ Umgebungsmodellservers ist es allerdings noch nicht möglich, dass Nexus-Anwendungen Objekte im KontextWiki anlegen, die dann über die KontextWiki-Anwendung wieder zugreifbar sind.

7.2.4 Fazit

Zusammenfassend kann man sagen, dass sich das Konzept eines Umgebungsmodellservers leicht auf bestehende Systeme übertragen lässt. Das Problem der Abbildung von Schemata bleibt natürlich bestehen, wobei die Struktur des Umgebungsmodells diese Aufgabe erleichtert: sehr allgemeine Objekttypen werden auf „obere“ Nexus-Objekttypen wie **SpatialObject** oder **MobileObject** abgebildet, während speziellere Typen entsprechend tiefer in der Hierarchie eingehängt werden können oder über ein erweitertes Schema.

Durch die Integrationsstudien hat sich zudem gezeigt, dass das Konzept eines generischen erweiterten Schemas sinnvoll ist: dieses schreibt nicht vor, welche Objekttypen und Attribute nun genau in einem Objekt enthalten sind, sondern überlässt es der Anwendung, die Daten zu verstehen. Damit sind zwar keine weiteren Funktionen durch die Plattform möglich (da diese das Schema nicht kennt), aber bestehende Anwendungen können gleichzeitig für Daten des Standard-Schemas die Plattform-Funktionen nutzen, während sie weiterhin Daten der bestehenden Lösung erhalten.

7.3 Zukünftige Erweiterungen

In diesem Kapitel werden Weiterentwicklungen des Umgebungsmodells vorgestellt, die geplant oder schon teilweise umgesetzt sind. Allerdings sind diese Konzepte häufig noch nicht implementiert oder durch Anwendungen evaluiert, so dass sie eher als Ausblick auf die Zukunft zu verstehen sind denn im Sinne einer Ergebnisdokumentation.

7.3.1 Metadaten

Metadaten sind allgemein „Daten über Daten“, sprich zusätzliche Informationen, welche die tatsächlich benötigten Daten beschreiben oder in einen größeren Zusammenhang setzen. Was in einem bestehenden System nun Daten und was Metadaten sind, hängt stark davon ab, was die eigentliche Funktion des Systems ist.

Im Datenbankbereich werden häufig die Schemainformationen als Metadaten bezeichnet, sprich die Tabellendefinitionen und Datenschemata. Diese Art von Metadaten unterstützt das Umgebungsmodell bereits: über die **schemaLocation**, eine URL, die in XML-Dateien angegeben sein sollte (und in der Implementierung der Nexus-Plattform auch ist), können jederzeit die zugehörigen Schemadateien (z.B. NSAS, NSAT,...) geladen und ausgewertet werden.

Im Bereich von Informationssystemen und digitalen Bibliotheken beschreiben Metadaten meist die Eigenschaften einer Informationsquelle, z.B. welche Genres von Büchern in einem bestimmten Katalog gespeichert sind. Diese Metadaten sind vergleichbar mit den Informationen, die ein Umgebungsmodellserver im räumlichen Verzeichnisdienst ablegt, denn sie beschreiben die Eigenschaften eines lokalen Umgebungsmodells: seine Ausdehnung (Dienstgebiet) und seine Inhalte (Objekttypen).

Für das Umgebungsmodell sollen nun zusätzlich zu den normalen Attributen und Objekten Metadaten gespeichert werden, welche die Umgebungsinformation selbst annotieren. Diese können vom System generiert oder übergeben werden (z.B. der Messzeitpunkt), technisch messbar sein (die Qualität eines Kamerabilds abhängig von den herrschenden Lichtverhältnissen), technische Einschränkungen darstellen (z.B. die bekannte Ungenauigkeit eines Lokalisierungssystems), die Herkunft der Daten beschreiben (z.B. Autor oder Zertifikate) oder die Kosten beschreiben, die bei der Erhebung oder Anfrage der Daten entstehen.

Solche Metadaten können dazu eingesetzt werden, verschiedene Fragestellungen zu beantworten:

- Verlässlichkeit: Wie sehr traue ich den Daten? Ist die Datenquelle vertrauenswürdig?
- Genauigkeit: Wie genau sind die Daten?
- Konsistenz: Passen verschiedene Datensätze überhaupt zusammen? Warum widersprechen sie sich?
- Alter: Wie aktuell sind die Daten?

In [HKN+05] wird ein erster Ansatz zur Modellierung von Metadaten im Umgebungsmodell vorgestellt. Abbildung 29 zeigt ein Beispiel zu Metadaten zu einem Objekt. Die Attribute **position** und **temperature** haben zusätzlich zu ihrem Wert (**value**) noch Informationen über ihre Genauigkeit bzw. ihren Messzeitpunkt. Für die Temperatur existieren zwei Werte, die zu unterschiedlichen Zeiten gemessen wurden. Zudem gibt es noch Metadaten, die sich auf das gesamte Objekt beziehen: Die Autorin heißt *Alice* und das Objekt wurde zu einem bestimmten Zeitpunkt angelegt.

In den aktuellen Schemata des Umgebungsmodells sind Metadaten bereits vorgesehen. Die Modellerierung hat jedoch Auswirkungen auf die Implementierung. So ist es nun vorgesehen, dass von einem Attribut mehrere Werte (mit unterschiedlichen Metadaten) existieren können. Wir

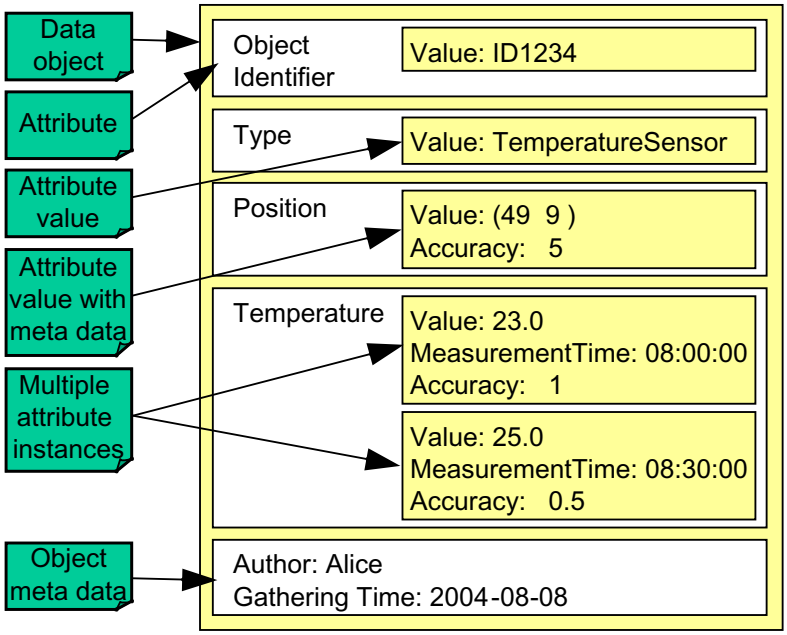

Abbildung 29: Metadaten zu einem Datenobjekt (aus [HKN+05])

sprechen dabei von Attributsinstanzen. Die Anfragesprache AWQL musste so modifiziert werden, dass sie neben der Auswahl von Objekten und Attributen nun auch die Filterung von den gewünschten Attributsinstanzen zulässt.

Auch die Implementierung eines Umgebungsmodellservers ist von dieser Änderung betroffen. Eine erste Implementierung wurde nun abgeschlossen, mit der mehrere Attributsinstanzen bei Positionsattributen und damit Positionshistorien gespeichert werden können. Damit ist ein erster Schritt die Richtung „temporale Erweiterung des Umgebungsmodell“ getan, mit dem nicht mehr nur der aktuelle Stand der Welt sondern auch vergangene Zustände (und zukünftige Prognosen) verwaltet werden können. Dies würde jedoch das Thema dieser Arbeit sprengen und wird deswegen nicht weiter behandelt.

7.3.2 Sensoren

Die Modellierung von Sensoren und Sensordaten ist bereits im aktuellen Umgebungsmodell vorgesehen (siehe Kapitel 6.7.5). Allerdings ist diese eher konzeptionell und nicht sonderlich detailliert: es sind nur wenige konkrete Sensoren tatsächlich im Schema definiert, da bisherige Anwendungen diese nicht benötigten.

Im Rahmen einer Arbeitsgruppe des Nexus-Projekts wurde eine Erweiterung des Umgebungsmodells erarbeitet [DKK+04], die eine Vielzahl möglicher Sensoren und deren Messungen (Sensordaten) modelliert. Dazu wurden verschiedene bestehende Modellierungsstandards untersucht, bei der sich eine sehr gute Eignung der Standards *SensorML* [SensorML] (für Sensoren) und *Observations and Measurements* [Observations] für die Anforderungen zukünftiger Anwendungen zeigte. Deswegen orientiert sich die Erweiterung des Umgebungsmodell stark an den oben genannten Standards, was eine Übersetzung in und aus dem Umgebungsmodell einfach macht. Damit können später Werkzeuge, die diese Standards verstehen, Daten aus dem Umgebungsmodell einfach verarbeiten.

Für die Verwaltung solcher Daten wird derzeit ein generischer Dienst entwickelt, der Daten von Sensoren aufnimmt und eventuell fusioniert, sowie ein Umgebungsmodellserver, der um eine Schnittstelle zu diesem Dienst erweitert ist und beliebige Sensorwerte und deren Historien verwalten kann.

7.3.3 Dienste

Die bisher vorgestelle Architektur unterstützt besonders datenorientierte ortsbezogene Anwendungen, die häufig dem Bereich *mobile computing* zuzuordnen sind. Mit einer Ausweitung der Anwendungsdomäne auf den Bereich des *ubiquitous computing* ist es sinnvoll, auch dort verbreitete Systemmodelle zu unterstützen. Viele dieser Anwendungen sind dienstorientiert: sie basieren darauf, verschiedene Dienste in der Umgebung (z.B. Videoprojektor, Telefon, Drucker) aufzufinden und dynamisch zu binden, um so dem Benutzer einen allgegenwärtigen und einheitlichen Zugriff zu ermöglichen.

Allgemein folgen dienstorientierte Architekturen dem Systemmodell *publish, find, bind:* ein Dienst wird zunächst in einem Dienstverzeichnis angemeldet, über den eine Anwendung passende Dienste finden kann. Schließlich nutzt die Anwendung die Beschreibung des Verzeichnisses, um sich an den Dienst zu binden und um ihn tatsächlich zu verwenden.

Bisherige Dienstverzeichnisse (z.B. [UDDI]) unterstützen keine ortsbasierten Anfragen nach Diensten (z.B. die Frage nach Druckern im Umkreis von 30 Metern). Hierzu kann das Umgebungsmodell verwendet werden. Es ist dort bereits vorgesehen, technische Objekte dort zu modellieren (siehe Kapitel 6.7.5). Das Modell soll nun um ein allgemeines Konzept für die Beschreibung von Diensten (zu denen z.B. auch Sensoren gehören können) erweitertet werden.

Dazu wurden bestehende Ansätze zur Dienstmodellierung untersucht. Für die Modellierung von Diensten, die über eine SOAP- bzw. Webschnittstelle verfügen, ist [WSDL] am besten geeignet, da es sich um einen Standard handelt, den viele Werkzeuge bereits unterstützen. Für hardwarenahe Dienste wie Peripherie-Geräte orientiert sich die Dienstbeschreibung am Standard *Universal Plug and Play* [UPnP].

Bei der Modellierung wurde auf ortsbezogene Dienste fokussiert. Solche Dienste haben einen Ort, an dem sie sich befinden (z.B. ein Drucker,...), und häufig auch ein begrenztes Gebiet, in dem sie sinnvoll nutzbar sind (z.B. ein Raum, in dem ein Lautsprecher zu hören ist). Nicht-ortsbezogene Dienste können weiterhin über existierende Ansätze verwendet werden.

Mit der Modellerweiterung für Dienste sollen mittelfristig drei verschiedene Systemmodelle unterstützt werden:

- Auffinden von Diensten, externes Binden: Eine Anwendung kann das Umgebungsmodell dazu nutzen, ortsbezogene Dienste zu finden, um diese dann über die Dienstbeschreibung extern zu binden und zu verwenden. Dabei ist es nicht nötig, dass die gesamte Dienstbeschreibung abfragbar ist. Vielmehr reichen wenige, relevante Attribute, die Anwendungen für die Auswahl von Diensten benötigen.
- Dienste als Aktoren: Anwendungen sollen Dienste auch direkt über das Umgebungsmodell verwenden können. Durch das Setzen und Ändern bestimmter Attribute werden Aktionen in der physischen Welt ausgelöst. In diesem Fall hat sich der Umgebungsmodellserver, der das Dienstobjekt verwaltet, an den Dienst gebunden und kann solche Änderungsaktionen an den Dienst weitergeben.
- Auffinden von Anwendungen und lokale Ausführung: Auch ortsbezogene Anwendungen können als Dienste angesehen werden. Das Umgebungsmodell soll das Auffinden von Anwendungen ermöglichen, die dann auf das Endgerät herunter geladen und dort ausgeführt werden können. Die Voraussetzung hierfür ist ein detailliertes Modell von SystemVoraussetzungen, ein Sicherheitskonzept, um böswilligen Code zu vermeiden, und eine Ausführungsumgebung auf dem Endgerät.

Abbildung 30 gibt einen Überblick über die neuen Dienstobjekttypen im Umgebungsmodell. Das **ServiceObject** repräsentiert einen ortsbezogenen Dienst. Die **serviceArea** ist das Gebiet, in dem der Dienst nutzbar ist. Sollte er überall einsetzbar sein, so wird das Attribut weggelassen. Die **serviceDescription** ist ein XML-Attribut und enthält die WSDL-Beschreibung des Dienstes. Dieser XML-Attributtyp wurde für die Dienstbeschreibungen hinzugefügt.

Dieser Ansatz bietet Kompatibilität mit den Web-Service-Standards: bestehende Werkzeuge, die WSDL-Beschreibungen verstehen, können so auch von Nexus-Anwendungen genutzt werden, um auf so beschriebene Dienste zuzugreifen.

supportedOperations ist eine Liste von Funktionen, die der Dienst ausführen kann: z.B. *play, stop* bei einem CD-Spieler. Das Attribut **performOperation** gibt an, welche Funktion der Dienst gerade ausführt. Wird dieses Attribut auf einen anderen Wert gesetzt so führt der Dienst eine neue Funktion aus. Es handelt sich also um einen Aktor. Darüber

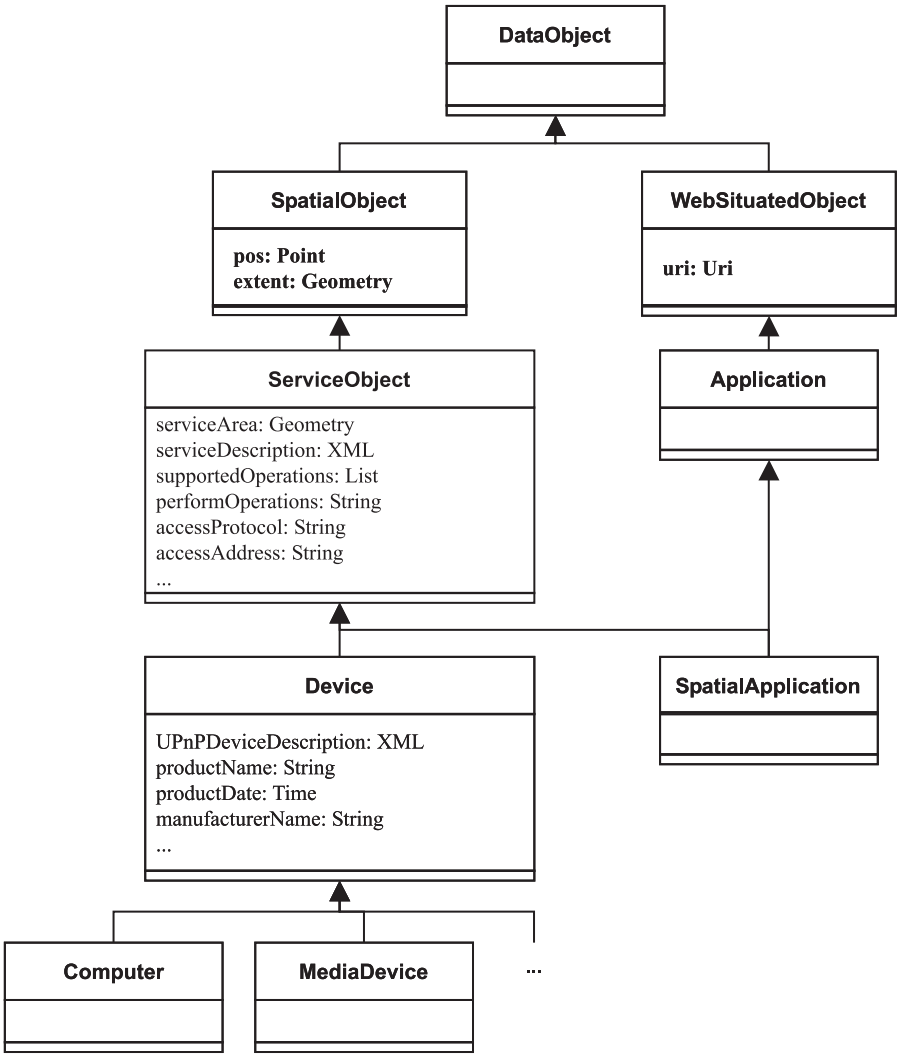

Abbildung 30: Dienst-Objekttypen im Umgebungsmodell (nach [Zha05])

hinaus hat das **serviceObject** weitere Attribute für unterstützte Kommunikationsprotokolle und Zustandsvariablen, auf die hier nicht weiter eingegangen wird.

Hardware-Dienste werden unter dem Objekttyp **Device** zusammengefasst. Ein **Device** hat eine **UPnPDeviceDescription**, in der die Information als XML-Attribut gespeichert ist, die für das Binden nach dem *UPnP*-Standard notwendig ist, sowie weitere Attribute, die das Gerät näher beschreiben und vor allem für Wartungsszenarien interessant sind. Unter dem Obertyp **Device** finden sich zahlreiche Geräte, die jeweils über für sie spezifische Attribute verfügen, um Anwendungen das Auffinden von

Farbdruckern mit Duplex-Einheiten, Computern mit einer bestimmten Prozessorleistung oder Scanner mit einer ausreichenden Auflösung zu ermöglichen.

Ortsbezogene Anwendungen sind im Modell bereits vorgesehen (**SpatialApplication**). Allerdings gibt es noch keine weitergehenden Überlegungen zu den notwendigen Attributen und der Systemumgebung, die ein lokales Herunterladen und Ausführen solcher Anwendungen ermöglicht.

Die Arbeiten an einem Nexus-Dienstmodell haben gezeigt, dass neben einem XML-Datentyp ein Bedarf an binären und strombasierten Datentypen besteht, mit denen beispielsweise ein Videostream übertragen werden kann. Hierfür sind das Umgebungsmodell und die Nexusplattform in seiner bisherigen Form nicht geeignet, da sie auf einer vollständigen Übertragung textbasierter Daten beruhen.

Die Arbeiten zur Dienstmodellierung ermöglichen die Entwicklung einer generischen mobilen Finder-Anwendung, die in ihrer Umgebung nach verfügbaren Diensten und räumlichen Anwendungen sucht und diese dem Benutzer anzeigt. Der Benutzer kann dann entscheiden, ob er die Anwendungen über das Umgebungsmodell installieren und nutzen will.

7.3.4 Komplexe Relationen

Bisher unterstützt das Umgebungsmodell nur einfache, nicht-räumliche Relationen (siehe Kapitel 6.2.9). In einer Arbeitsgruppe wurde ein Konzept für komplexe Relationen entwickelt und eine Modellerweiterung dafür entwickelt [DDG+04]. Komplexe Relationen werden als *heavyweight* (schwergewichtig) bezeichnet, nach der Definition von [OGC99], im Gegensatz zu den einfachen Relationen, die dort als *lightweight relations* bezeichnet werden.

Auf die Vorteile von einfachen Relationen wurde bereits in Kapitel 6.2.9 eingegangen. Komplexe Relationen sind eigenständige Objekte, die Referenzen auf die Objekte enthalten, die sie in Beziehung setzen. Die Vorteile dieses Ansatzes sind:

- Flexibilität: Es können Beziehungen zwischen beliebigen Objekten definiert werden.
- Restriktionen: Es können Einschränkungen definiert werden, wie oft Objekte an der Relation teilnehmen können.

- Stelligkeit: Für Beziehungen, an denen mehr als zwei Objekte teilnehmen sollen, sind komplexe Relationen zwingend notwendig.
- Unabhängigkeit: Komplexe Relationen können unabhängig von den teilnehmenden Objekten gespeichert werden. Damit muss ein Objekt nicht geändert werden, wenn eine Relation zu ihm eingerichtet wird.
- Typisierung: Die einfachen Relationen aus dem Umgebungsmodell bestehen aus einer NOL als Referenz, die auf Objekte beliebigen Typs verweisen können. In einem komplexen Relationsobjekt kann Information darüber abgelegt werden, welche Objekttypen an der Relation überhaupt teilnehmen dürfen.

Mit der Modellerweiterung sollen verschiedene Relationstypen unterstützt werden, darunter räumliche (z.B. Distanz, Richtung, Topologie), thematische (z.B. Kategorien), zeitliche und Relationen zwischen Mehrfachrepräsentationen. Erste Ideen dazu finden sich in [DDG+04].

Wie in Kapitel 6.2.9 bereits angedeutet, gibt es grundsätzliche Probleme mit der Verwaltung komplexer Relationen in der Nexusplattform. Es gibt zwei Möglichkeiten, die Relationsobjekte zu modellieren, mit oder ohne Ortsbezug:

- mit Ortsbezug: Das Relationsobjekt erhält einen Ortsbezug durch die teilnehmenden Objekte, in dem ein räumliches Attribut (**extent**) aus den Positionen der Objekte gebildet wird. Bei weit auseinanderliegenden Objekten wird dieses Attribut sehr ausgedehnt, was eine effiziente Verwaltung erschwert (ein Umgebungsmodellserver bräuchte ein sehr großes Dienstgebiet, um solche Relationen zu verwalten). Zudem gibt es Probleme bei der Verwaltung von Relationen, an denen mobile Objekte teilnehmen.
- ohne Ortsbezug: Der Relationenobjekttyp erbt direkt von **NexusDataObject**. Nun ist das Problem, dass die Relationenobjekte nicht über die Plattform auffindbar sind, sofern ihre **NOL** nicht bekannt ist, da sie über räumliche Anfragen nicht gefunden werden.

Momentan hat man sich für die zweite Lösung entschieden und das Problem der Auffindbarkeit durch Rückwärtsreferenzen gelöst, die in den teilnehmenden Objekten einer Relation gespeichert werden. Dabei handelt es sich um technisch generierte Attribute, die nicht von Anwendungen geändert werden können. Stattdessen sorgen die

Umgebungsmodellserver untereinander dafür, dass die Rückwärtsreferenzen (vom teilnehmenden Objekt zum Relationenobjekt) aktuell gehalten werden.

Diese Lösung wirft erneut ein Problem auf: was passiert, wenn ein Umgebungsmodellserver sich weigert, die Rückwärtsreferenz zu einem fremden Relationenobjekt zu speichern? Hierfür werden zwei Lösungen vorgeschlagen: entweder können Objekte solcher „unkooperativer“ Umgebungsmodellserver nicht an komplexen Relationen teilnehmen, oder es wird in Kauf genommen, dass nicht alle teilnehmenden Objekte eine Rückwärtsreferenz speichern auf Kosten der Auffindbarkeit des Relationsobjekts (das dann über Anfragen, die nur „unkooperative“ Teilnehmerobjekte auswählt, nicht gefunden werden kann). In einem ersten Schritt wird nun eine Lösung implementiert, die von kooperativen Umgebungsmodellservern ausgeht.

Allgemein ist zu sagen, dass das Verwalten und der Nutzen komplexer Relationen im Umgebungsmodell ein weiterführendes Thema ist, das noch deutlichen Forschungsbedarf darstellt.

7.3.5 Die dritte Dimension (3D)

Als die grundlegenden Konzepte des Umgebungsmodells definiert wurden, existierte kein allgemeiner Standard zur Modellierung dreidimensionaler Daten. Mit der *Simple Feature Specification* [SFS] stand dagegen eine Spezifikation zweidimensionaler geometrischer Primitive samt Prädikate zur Verfügung, für die es mit GML in der Version 2.0 [GML] auch eine XML-Repräsentation gab, die einfach in die bestehenden Schemata integriert werden konnte. Die ortsbezogenen Anwendungen, die zu Beginn mit Modell und Plattform unterstützt werden sollten, kamen auch mit zweidimensionalen Daten aus. Zudem ist die Präsentation dreidimensionaler Daten schwierig: herkömmliche Bildschirme unterstützen nur zweidimensionale Daten, so dass dreidimensionale Daten stets auf zwei Dimensionen projeziert werden müssen.

Inzwischen gibt es jedoch Bestrebungen, weitere Anwendungen durch das Modell zu unterstützen, z.B. die Navigation blinder Menschen in einem Gebäude oder die Produktion in einem Fabrikgebäude, bei der durch eine *Augmented Reality*-Präsentation für Arbeiter Zusatzinformationen zu Werkzeugen eingeblendet werden. Diese Anwendungen benötigen dreidimensionale Daten. Dazu kommt, dass in der Version 3.0 von

GML auch solche Daten unterstützt werden sollen. Deswegen wird derzeit untersucht, wie das Umgebungsmodell durch eine dritte Dimension erweitert werden kann.

Bis zur vollen Unterstützung dreidimensionaler Daten gibt es verschiedene Zwischenstufen:

- 2D mit symbolischer Höhenangabe: Für die Modellierung mehrstöckiger Gebäude ist eine reine zweidimensionale Darstellung nicht ausreichend. In einem ersten Schritt wurde deswegen das **Room**-Objekt mit einem Attribut **level** ergänzt, um das Stockwerk anzugeben, in dem sich der Raum befindet. Damit sind Anfragen möglich, die nur bestimmte Ebenen eines Raums betreffen. Für alle anderen Objekte ist diese Information allerdings nicht vorhanden und kann allenfalls durch Relationen (z.B. **partOf**) genutzt werden. Auch kann nur innerhalb eines Gebäudes entschieden werden, ob ein Raum über einem anderen liegt.
- 2,5D: Die Geometrien werden zweidimensional gespeichert, dazu kommt eine geometrische Höhenangabe. Damit sind bereits Anfragen über mehrere Gebäude und auch beliebige Objekte möglich. Die Höhenangabe wird jedoch in den räumlichen Prädikaten (**within, overlaps**) nicht berücksichtigt. Eine Filterung auf die Höhe muss also über eine eigene Restriktion formuliert werden.
- 3D-Daten ohne Prädikate: Ein weiterer Schritt ist es, alle räumlichen Objekte mit einem zusätzlichen Attribut auszustatten, das dreidimensionale Daten (z.B. im GML 3.0-Standard) enthält. Damit können bereits dreidimensionale Modelle gespeichert werden. Die Objektauswahl erfolgt allerdings weiterhin über die zweidimensionalen Attribute und Prädikate. Eine Anwendung – oder ein Mehrwertdienst – kann jedoch die dreidimensionalen Attribute auswerten und dann auf die dritte Dimension filtern.
- volle 3D-Unterstützung: Fernziel ist die volle Unterstützung dreidimensionaler Modelle. Dies geht allerdings nicht ohne die räumlichen Prädikate in ihrer Implementierung zu ändern. Je nach Aufkommen der dreidimensionalen Daten kann es auch notwendig werden, den räumlichen Verzeichnisdienst um die dritte Dimension zu erweitern. Dies ist jedoch nicht sehr wahrscheinlich, da es vermutlich selten der Fall sein wird, dass viele Umgebungsmodellserver das gleiche Gebiet,

aber in verschiedener Höhe abdecken, und die Selektivität bei einer rein zweidimensionalen Serverauswahl gut genug sein dürfte.

Morpheus: There is a difference between knowing the path and walking the path. (The Matrix)

Kapitel 8: Anwendungsentwicklung

Wie können nun mit Modell und Nexus-Plattform Anwendungen entwickelt werden? In diesem Kapitel wird ein allgemeines Vorgehen vorgestellt, das auf die Besonderheiten mobiler, ortsbezogener Anwendungen eingeht. Danach werden verschiedene konkrete Anwendungen vorgestellt, die zum Teil ohne, zum Teil mit Unterstützung des Umgebungsmodells und der Nexus-Plattform entwickelt wurden und dadurch aufgezeigt, wie sich die Ergebnisse dieser Arbeit im konkreten Einsatz bewähren. Abschließend wird ein allgemeiner Entwurf für kontextbezogene Anwendungen vorgestellt.

8.1 Allgemeines Vorgehen

Die Entwicklung mobiler, ortsbezogener Anwendungen unterscheidet sich grundsätzlich nicht von der allgemeinen Entwicklung von Softwaresystemen: es müssen Anforderungen erhoben und spezifiziert werden, die Systemarchitektur und eine geeignete Aufteilung in Module muss entworfen werden, diese müssen implementiert, integriert und getestet werden. Schließlich wird das fertige System in seiner Produktionsumgebung bereitgestellt und muss dort gewartet werden (vgl. z.B. [Som95]).

Es gibt jedoch einige Besonderheiten, in denen sich mobile, ortsbezogene Anwendungen von herkömmlichen Anwendungen unterscheiden, von denen an dieser Stelle die wichtigsten erwähnt werden sollen:

- Heterogenität der technischen Infrastruktur bzw. der Endgeräte: während herkömmliche Anwendungen in der Regel in einer standardisierten Systemumgebung ablaufen, gilt dies nicht für ortsbezogene Anwendungen. Es gibt eine große Auswahl mobiler Endgeräte mit unterschiedlicher Leistung (von Mobiltelefonen über persönliche digitale Assistenten (PDA) bis hin zu tragbaren Computern), drahtloser

Kommunikationsmöglichkeiten mit unterschiedlicher Verfügbarkeit und Bandbreite (GPRS, Bluetooth, UMTS, WLAN,...) und Sensoren mit unterschiedlicher Verfügbarkeit und Genauigkeit (Infrarot, GPS,...). Auch Anwendungen, die in instrumentierten Umgebungen ablaufen, haben es mit einer ähnlich heterogenen Systemlandschaft zu tun. Die technische Infrastruktur hat starken Einfluss auf die Möglichkeit und Fähigkeit der Anwendung, mit dem Benutzer zu interagieren, mit anderen Systemen zu kommunizieren, ihre Berechungen effizient durchzuführen und ihre Daten zu speichern. Dazu kommt, dass die technischen Möglichkeiten sich in hohem Tempo verändern. Deswegen kann man sich keine langen Entwicklungszyklen leisten und darf Anwendungen auch nicht zu sehr auf ein bestimmtes Endgerät hin optimieren. Es könnte veraltet sein, wenn die Anwendung auf den Markt kommen soll.

- Verfügbarkeit von Umgebungsdaten: Je größer der räumliche Bereich ist, in dem die Anwendung laufen soll, um so schwieriger wird es, sicher zu stellen, dass bestimmte Umgebungsdaten verfügbar sind, und auch in der benötigten Vollständigkeit, Genauigkeit oder Detaillierung. Es liegt also in der Natur ortsbezogener Anwendungen, dass sie bezüglich der verfügbaren Daten eine gewisse Flexibilität aufweisen müssen.
- Unerfahrene Benutzer: Da mobile, ortsbezogene Anwendungen noch relativ neu sind, haben die meisten Benutzer keine Erfahrungen mit den Endgeräten oder den Anwendungen selbst. Es müssen neue Interaktionsformen gefunden werden, welche die Benutzer auch erst erlernen müssen. Auch dies erfordert schnelle Entwicklungszyklen und frühe Einbindung der Benutzer in den Entwicklungsprozess (z.B. über Prototypen).
- Neue Möglichkeiten in der Bereitsstellung: Neben herkömmlicher Möglichkeiten, Software auf die Systeminfrastruktur des Kunden zu installieren (z.B. über Installations-CDs oder aus dem Internet) sind für mobile ortsbezogene Anwendungen auch neue Wege denkbar. So könnte das räumliche Gebiet, für das die Anwendung entwickelt wurde, als Auswahlkriterium dienen: ein Anwendungs-Finder-System auf dem mobilen Endgerät könnte nach geeigneten Anwendungen in der Nähe Ausschau halten und diese gegebenenfalls dem Benutzer anbieten.

Im Folgenden soll nun näher auf die Entwicklung ortsbezogener Anwendungen unter Berücksichtigung der Ergebnisse dieser Arbeit eingegangen werden. Dabei wird zunächst eine grobe Kategorisierung bezüglich wichtiger Entwurfsentscheidungen (die häufig von der Systeminfrastruktur diktiert werden) vorgenommen, anhand dessen konkrete Anwendungen vorgestellt und eingeordnet werden. Abschließend wird daraus ein genereller Entwurf einer Anwendung abgeleitet.

8.1.1 Kategorisierung

Tabelle 6 gibt einen Überblick über wichtige Funktionen mobiler, ortsbezogener Anwendungen und den daraus resultierenden Entwurfsentscheidungen.

Tabelle 6: Kategorisierung von Entwurfsentscheidungen

Aspekt	mögliche Entscheidungen
Mobilität	stationär, begrenzt, unbegrenzt
Datenspeicherung	lokal, entfernt, hybrid
Drahtlose Kommunikation	keine, langsam, schnell, variabel
Erreichbarkeit	nie, ständig, variabel
Datenverarbeitung	einfach, komplex
Lokalisierung	Benutzer, lokal, entfernt, hybrid
Umgebungsmodellschema	kein, standard, erweitert
Nutzung des Umgebungsmodells	keine, lesen, schreiben, hybrid

Mobilität

Die Mobilität der Benutzer bestimmt das geplante Einsatzgebiet der Anwendung. Stationäre Anwendungen arbeiten nur an einem bestimmten Punkt (z.B. eine Informations-Station am Eingang eines Gebäudes, die einem vorübergehenden Benutzer Pläne anzeigt und Navigationshinweise gibt). Begrenzte Mobilität liegt dann vor, wenn die Anwendung für ein bestimmtes Gebiet konzipiert wurde (z.B. für eine instrumentierte Umgebung oder eine Stadt), während bei unbegrenzter Mobilität die Anwendung potentiell global eingesetzt wird.

Der Grad an Mobilität hat Auswirkungen auf einige der folgenden Aspekte, die dann jeweils dort erläutert werden.

Datenspeicherung

Eine Anwendung kann die benötigten Umgebungsmodelldaten lokal auf dem Endgerät oder entfernt in der Infrastruktur speichern, oder sie zwischen lokalem und entferntem Speicher aufteilen. Lokale Speicherung hat den Vorteil, dass zum Zugriff auf die Daten keine Kommunikationsverbindung aufgebaut werden muss, und somit Zeit und Kosten gespart werden können. Allerdings können die Daten u.U. schnell veralten, und die Datenmenge kann je nach Anwendungsfall zu groß werden, als dass sie noch mit den beschränkten Ressourcen des Endgeräts verwaltet werden kann. Als Lösung bietet sich zum einen eine hybride Verteilung an, bei der sich häufig ändernde Daten aus der Infrastruktur geholt werden, während eher statische Daten lokal vorgehalten werden. Zum anderen können auch Caching-Konzepte eingesetzt werden, die den besonderen Charakteristika ortsbezogener Daten Rechnung tragen (siehe z.B. [ZXL02]).

Die Wahl der Datenspeicherung ist auch abhängig vom Grad der Mobilität: wenn die Anwendung nur ein kleines Gebiet abdeckt, ist u.U. auch die Datenmenge überschaubar. Das Gerät kann dann ein für die Anwendung optimiertes Modell seiner Umgebung lokal vorhalten und gegebenenfalls durch aktuelle Sensordaten ergänzen. Je größer das abgedeckte Gebiet jedoch wird, um so mehr ist die Anwendung auf externe Daten angewiesen und wird diese bei Bedarf nachladen.

Drahtlose Kommunikation

Der Durchsatz der drahtlose Kommunikationsverbindung zur Infrastruktur kann stark variieren und ist unter anderem abhängig von der verwendeten Technik, der Auslastung oder der physischen Entfernung zum Zugangspunkt. Wenn die Anwendung über mehrere Kommunikationstechniken verfügt (z.B. WLAN und GPRS) kann sie auch variabel sein: die Anwendung wählt dann aus den verfügbaren Kommunikationsmöglichkeiten die für sie geeigneteste aus.

Auch dieser Aspekt ist abhängig von der Mobilität: innerhalb eines begrenzten Gebiets kann eine bestimmte Technologie oder auch eine Dienstgüte hergestellt und auch vorausgesetzt werden. Bei unbegrenztem Einsatz muss sich die Anwendung auf wechselnde Kommunikationsbedingungen bis hin zum Wegfall der Verbindung gefasst machen.

Erreichbarkeit

Dieser Aspekt bezieht sich auf die Erreichbarkeit der Anwendung durch die Plattform. Ständige Erreichbarkeit bedeutet, dass die Plattform Systemnachrichten an die Anwendung ausliefern kann (z.B. geographische Nachrichten oder Ereignisse). Dies wird unmöglich, wenn die Anwendung anonym bleibt und zwischen ihren Informationsanfragen die Kommunikationsverbindung trennt. Es sind auch hier variable Formen denkbar, bei denen die Anwendung nur zu bestimmten Zeiten oder in bestimmten Gegenden erreichbar ist.

Auch dieser Aspekt wird von der Mobilität beeinflusst. Ständige Erreichbarkeit ist bei unbegrenztem Einsatz schwer zu erreichen: selbst der weitverbreitete Mobilfunk kann dies heutzutage noch nicht leisten. Für ein begrenztes Gebiet können jedoch Anwendungen entworfen werden, die von ständiger Erreichbarkeit ausgehen.

Datenverarbeitung

Die Möglichkeiten der Datenverarbeitung werden stark durch die technische Infrastruktur bestimmt. Gerade bei mobilen Endgeräten müssen hier häufig Abstriche gemacht werden. Komplexe Datenverarbeitung benötigt leistungsstarke Prozessoren, während einfache Algorithmen bereits auf schwächeren Systemen ausgeführt werden können.

Wenn der Einsatz der Anwendung auf einem leistungsschwachen Gerät geplant ist und trotzdem komplexe Berechnungen notwendig sind, können solche Funktionen u.U. in die Infrastruktur ausgelagert werden. Die Nexus-Plattform bietet hierfür die Möglichkeit von Mehrwertdiensten (siehe Kapitel 5.2.4).

Lokalisierung

Lokalisierung ist die Bestimmung der Position eines Objekts, z.B. des Benutzers. Dies ist für ortsbezogene Anwendungen essentiell, da sie nur so in der Lage sind, ihre Funktionen dem Ort des Benutzers anzupassen. Hier bedeutet die Entwurfsentscheidung „Benutzer“, dass der Benutzer selbst wissen muss, wo er ist, und dies der Anwendung angibt (z.B. durch Klick auf eine Karte oder durch Texteingabe). Wird die Lokalisierung durch Sensorik durchgeführt, so kann sie entweder lokal (durch

Sensoren auf dem Endgerät, z.B. GPS), entfernt (durch Sensoren in der Infrastruktur, z.B. Kameras) oder hybrid (Kombination verschiedener Verfahren) durchgeführt werden.

Umgebungsmodellschema

Das Umgebungsmodell wurde so entworfen, dass es die von ortsbezogenen Anwendungen am häufigsten benötigten Objekttypen bereits im Standard-Schema spezifiziert. Trotzdem kann es für speziellere Anwendungen notwendig sein, eine Erweiterung zu definieren.

Auch hier spielt die geplante Mobilität eine Rolle: es ist davon auszugehen, dass viele Umgebungsmodellserver Objekte des Standard-Schemas anbieten, während Objekte erweiterter Schemata auf weniger Servern zu finden sind. Soll die Anwendung unbegrenzt eingesetzt werden, und benötigt sie Daten in einem erweiterten Schema, so könnte sie unterwegs Probleme mit ihrer Datenversorgung bekommen. Hierzu ist in der Entwicklungsphase eine Analyse sinnvoll, welche Umgebungsmodellserver im geplanten Einsatzgebiet bereits vorhanden sind und Objekte welcher Schemata diese anbieten. Dazu kann der räumliche Verzeichnisdienst (Kapitel 5.2.2) verwendet werden.

Nutzung des Umgebungsmodells

Schließlich ist zu entscheiden, ob die Anwendung nur lesenden Zugriff auf das Umgebungsmodell benötigt oder auch Objekte in das Modell einbringen und ändern will.

Auch hier ist die gewünschte Mobilität zu berücksichtigen und gegebenenfalls eine Analyse vorzunehmen, ob das gewünschte Einsatzgebiet von Umgebungsmodellservern abgedeckt wird, die der Anwendung auch einen Schreibzugriff auf ihr lokales Umgebungsmodell erlauben. Sollte dies nicht der Fall sein, müsste der Anbieter der Anwendungen eigene Server aufstellen und beim räumlichen Verzeichnisdienst anmelden, die dann die neuen oder geänderten Objekte der Anwendung speichern können.

Im Folgenden werden nun konkrete Anwendungsentwicklungen im Hinblick auf diese Aspekte vorgestellt und aufgezeigt, wie diese die Ideen dieser Arbeit verwenden.

8.2 Fakultätsinformationssystem

Die erste Anwendung fällt in das Gebiet „Ortsbezogene Informationssysteme“ (siehe Kapitel 4.1.2). In einem begrenzten Gebiet (dem Gebäude der Fakultät Informatik) sollen Informationen zu Räumen, Mitarbeiten, Lehrveranstaltungen, etc. ortsbezogen auf einem mobilen Endgerät angezeigt werden.

Um das Anwendungsgebiet besser kennen zu lernen, wurde zunächst eine einfache Anwendung entwickelt, die noch ohne das Umgebungsmodell und die Nexus-Plattform auskommt (lokale Datenspeicherung). Später wurde diese durch einen Umgebungsmodellserver ergänzt, der weitergehende Funktionen ermöglicht.

Tabelle 7 gibt einen Überblick über die Entwurfsentscheidungen dieser Anwendungen.

Tabelle 7: Entwurfsentscheidungen Fakultätsinformationssystem

Aspekt	einfache Anwendung	erweiterte Anwendung
Mobilität	begrenzt	begrenzt
Datenspeicherung	lokal	hybrid
Drahtlose Kommunikation	keine	variabel
Erreichbarkeit	nie	variabel
Datenverarbeitung	einfach	einfach
Lokalisierung	Benutzer	lokal
Umgebungsmodellschema	kein	standard
Nutzung des Umgebungsmodells	kein	lesen

Das einfache Fakultätsinformationssystem wurde im Jahr 2000 für ein kleines, mobiles Endgerät, einem sogenannten Visor entwickelt [Grz00]. Der Visor ist ein persönlicher digitaler Assistent (PDA), der damals über den Prozessor MC68328 mit 16 MHz Taktfrequenz, 8 Megabyte Speicher, einen Infrarot-Port sowie einem Schwarz-weiß-Bildschirm mit 160x160 Bildpunkten verfügte.

Ziel der Anwendungsentwicklung war es, mit einfachen Mitteln eine erste ortsbasierte Anwendung zu entwickeln, um Anforderungen und Erfahrungen für die Entwicklung des Umgebungsmodells und der Nexus-Plattform zu sammeln.

Es wurden dabei weitgehend vorhandene Daten eingesetzt: die Telefon- und Raumliste der Informatik-Mitarbeiter sowie Raumpläne im Vektordatenformat. Diese wurden über ein einfaches Raummodell (alle Räume sind rechteckig und liegen in einem lokalen Koordinatensystem) räumlich miteinander verknüpft.

Die Daten sind lokal auf dem Endgerät gespeichert, in einer mobilen Datenbank (DB2Everywhere). Damit können bei einem sehr geringen Speicherverbrauch bereits relationale Daten gespeichert und über eine eingeschränkte SQL-Schnittstelle abgefragt werden. Außerdem gibt es die Möglichkeit, diese Daten über einen Dienst mit einer stationären Datenbank zu synchronisieren. Durch diese Lösung war es nicht nötig, dass die Anwendung drahtlos kommuniziert: die Daten wurden zu Beginn auf das Gerät gespielt und verblieben während der Anwendung dort.

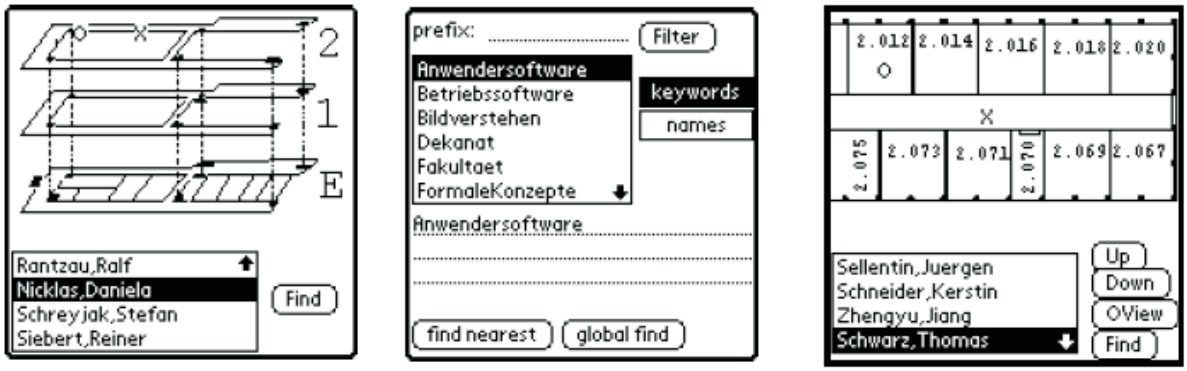

Abbildung 31: Bildschirmabzüge des Fakultätsinformationssystems

Abbildung 31 zeigt drei verschiedene Bildschirmabzüge der Anwendung, anhand derer sich die Funktionen erläutern lassen. Die Übersichtskarte zeigt die drei Stockwerke des Gebäudes an. Durch Klick auf einen Gebäudebereich wird die entsprechende Detailkarte angezeigt (rechts), über die auch die Lokalisierung stattfindet: der Benutzer gibt durch einen Klick an, wo er sich befindet. Die räumliche Suche (Abbildung 31, Mitte) ermöglicht das Auffinden des nächstgelegenen (*find nearest*) Mitarbeiters,

der über einen Suchtext (*prefix*) oder über die Auswahl einer bestimmten Gruppe (*keywords)* weiter eingeschränkt werden kann. Außerdem kann auch der gesamte Datenbestand durchsucht werden (*global find*). Auch diese Ergebnisse werden in einer Detailkarte angezeigt (rechts).

Die Vorteile eines solchen einfachen Anwendungsentwurfs liegen auf der Hand:

- der Benutzer hat stets alle Daten dabei, es ist keine drahtlose Kommunikation nötig, um sie auf das Endgerät zu laden
- es sind keine Sensoren für die Lokalisierung des Benutzers nötig

Allerdings hat die Lösung auch viele Einschränkungen:

- Die Daten können zur Laufzeit nicht aktualisiert werden (weder die Kartendaten noch die Mitarbeiterdaten).
- Keine automatische Lokalisierung: der Klick auf die Karte erfordert vom Benutzer zu wissen, wo er ist. Davon ist nicht immer auszugehen.
- Die Daten können nur schwer von anderen Anwendungen verwendet werden, da sie für den speziellen Anwendungsfall organisiert wurden.
- Der Anwendungsfall ist insgesamt sehr eingeschränkt. Erweiterte Funktionen wie Navigation, andere Ausgaben (z.B. Audio) sowie die Portierung auf andere Endgeräte mit anderen Bildschirmgrößen sind sehr aufwändig.

Insgesamt ist zu sagen, dass einfache Anwendungen auch ohne die Verwendung des Umgebungsmodells und der System-Plattform entwickelt werden können. Jedoch kann man durch den Ansatz eines Umgebungsmodells bereits in diesem einfachen Fall profitieren, ohne die eigentliche Anwendung wesentlich ändern zu müssen:

- Die lokalen Daten könnten jeweils beim Betreten des Gebäudes komplett über eine Anfrage im Umgebungsmodell zusammengestellt und in den lokalen Speicher des Endgeräts übertragen werden. Die Karten könnten vom Kartendienst in der benötigten Größe berechnet werden. Damit wären Karten und Daten zumindest tagesaktuell, was für Mitarbeiterdaten und Raumdaten ausreichen dürfte. Die Übertragung der Daten könnte an speziellen Ladestationen für das mobile Endgerät geschehen. Eine drahtlose Kommunikation wäre damit nicht notwendig.
- Wenn die Anwendung auf anderen Endgeräten laufen soll, würden die Karten in einer anderen Größe berechnet.

- Für die automatisierte Lokalisierung könnte die Infrarot-Schnittstelle verwendet werden. Voraussetzung ist, dass das Gebäude mit Infrarot-Emittern ausgestattet wird, die in einem Bereich von wenigen Metern eine ID ausstrahlen. Empfängt das Endgerät eine solche ID, so kann es über eine Abbildungsfunktion auf seine Position schließen. Auch die Abbildung von ID auf Positionen könnte aus dem Umgebungsmodell auf das Endgerät geladen werden.

Eine erweiterte Anwendung würde also eine hybride Datenspeicherung verwenden. Sofern das Endgerät über drahtlose Kommunikation verfügt, könnten Daten dynamisch nachgeladen werden. In diesem Fall wäre es für die Plattform auch erreichbar (auch wenn das für den Funktionsumfang nicht notwendig ist). Die Lokalisierung über Infrarot findet nun lokal auf dem Endgerät statt.

Für solche Anwendungen reicht das Standard-Schema des Umgebungsmodells aus, das bereits Personen, Räume und Infrarot-Emitter enthält. Es ist also damit zu rechnen, dass durch die Verwendung des Umgebungsmodells die erweiterte Anwendung auch in anderen Gebäuden funktionieren würde.

8.3 *NexusScout:* ein mobiles Stadtinformationssystem

Der *NexusScout* ist ein mobiles, ortsbasiertes Stadtinformationssystem, fällt also in das gleiche Anwendungsgebiet wie das Fakultätsinformationssystem. Neben den üblichen Funktionen, die solche Anwendungen bieten, implementiert er noch weitere, fortgeschrittene Anwendungsfälle, die z.T. erst durch den Einsatz der Nexus-Plattform und des Umgebungsmodells möglich werden [NGS+01].

Abbildung 32 zeigt einen Bildschirmabzug des *NexusScout*, auf dem einige der nun folgenden Anwendungsfälle zu erkennen sind.

8.3.1 Klassische Anwendungsfälle

Wie auch andere Vertreter seines Anwendungsgebiets bietet der *NexusScout* folgende Anwendungsfälle:

- Anzeigen einer Umgebungskarte des Benutzers
- Einzeichnen der eigenen Position in diese Karte (der Punkt in Abbildung 32)

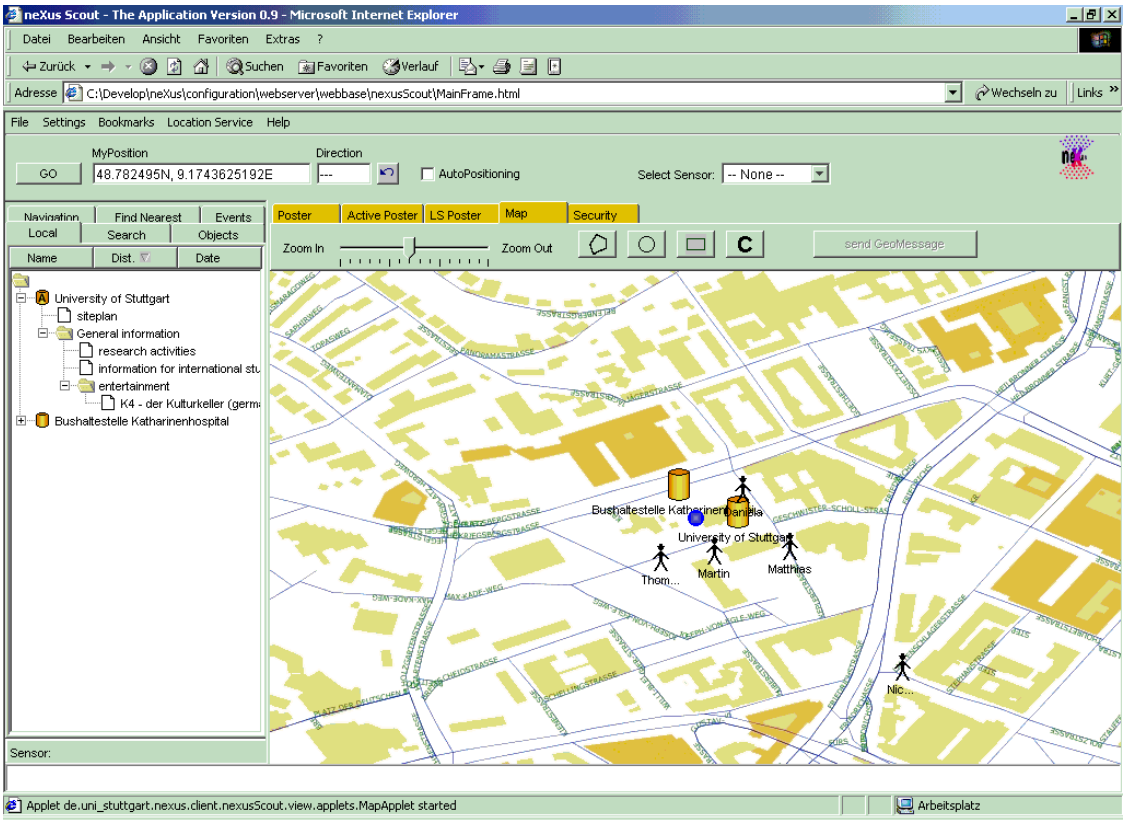

Abbildung 32: Bildschirmabzug des *NexusScout* [NGS03]

- Suche nach und Anzeigen von wichtigen Orten (*POIs – Points of Interest),* z.B. Restaurants
- Verknüpfung von *POIs* mit Webseiten (über Virtuelle Litfaßsäulen, die Tonnen in Abbildung 32)
- Benutzer-Lokalisierung über lokale GPS-Sensoren
- Navigation: der Benutzer kann sich eine Route von einem Start- zu einem Zielort berechnen und anzeigen lassen.

8.3.2 Erweiterte Anwendungsfälle

Darüber hinaus werden diese Anwendungsfälle jedoch durch Zusatzfunktionalität aufgewertet und durch weitere Anwendungsfälle ergänzt:

- Dynamisch erzeugte Karte: Die Karten des *NexusScout* werden dynamisch aus Daten des Umgebungsmodells berechnet. Dadurch können sie in verschiedenen Auflösungen und Darstellungsformen erzeugt werden und enthalten stets die aktuell im Umgebungsmodell enthaltenen Informationen

- Offene Verknüpfung von Webinhalten: Die Verknüpfung von *POIs* zu Webseiten ist nicht durch einen Anbieter vorgegeben. Durch die Offenheit der Nexus-Plattform können verschiedene Anbieter virtuelle Litfaßsäulen definieren und mit Webinhalten verknüpfen. Auch diese Verknüpfung wird vom *NexusScout* dynamisch aus dem Umgebungsmodell geladen.
- Andere Sensoren zur Lokalisierung: Da die Nexus-Plattform und das Umgebungsmodel unabhängig von einem konkreten Koordinatensystem ist, können auch andere Sensoren zur Lokalisierung eingesetzt werden. Der *NexusScout* unterstützt neben GPS auch Infrarot sowie die Lokalisierung durch den Benutzer.
- Modell-übergreifende Navigation: Der Mehrwertdienst Navigation berechnet eine Route nicht auf einem einzigen Modell, sondern verwendet dazu die integrierten Daten des Umgebungsmodells. Dadurch kann er auch über Modellgrenzen hinweg Routen berechnen.
- Andere mobile Objekte: Da das Umgebungsmodell nicht nur stationäre, sondern auch mobile Objekte enthält, kann der *NexusScout* ausgewählte andere Benutzer in die Karte einzeichnen, z.B. die Mitglieder einer Reisegruppe (die Strichmännchen in Abbildung 32).
- Räumliche Events: Auch die Position des Benutzers selbst wird im Umgebungsmodell gespeichert. Dadurch können in der Nexus-Plattform räumliche Ereignisse registriert und beobachtet werden, z.B. wenn der Benutzer ein bestimmtes Gebiet betritt oder einen anderen Benutzer trifft.
- Geographische Nachrichten: Das Umgebungsmodell kann auch dazu verwendet werden, eine Nachricht nicht an einen bestimmten Adressaten, sondern an alle Personen, die sich in einem bestimmten Gebiet aufhalten, zu senden.

8.3.3 Entwurfsentscheidungen

Tabelle 8 gibt einen Überblick über die Entwurfsentscheidungen, die dem *NexusScout* zugrunde liegen.

Der Einsatz der Anwendung ist grundsätzlich unbegrenzt: sofern Daten vorhanden sind, dürfte er in jeder Umgebung funktionieren. Allerdings erfordert das Laden von Karten und großer Webseiten eine schnelle Kommunikationsverbindung, was den Einsatz in schwach abgedeckten Gebieten erschwert.

Tabelle 8: Entwurfsentscheidungen *NexusScout*

Aspekt	*NexusScout*
Mobilität	(unbegrenzt)
Datenspeicherung	entfernt
Drahtlose Kommunikation	schnell (variabel)
Erreichbarkeit	variabel
Datenverarbeitung	einfach
Lokalisierung	lokal, Benutzer
Umgebungsmodellschema	standard
Nutzung des Umgebungsmodells	lesend

Die Datenspeicherung erfolgt entfernt im Umgebungsmodell, allenfalls Karten und Zwischenergebnisse werden auf dem Endgerät temporär gespeichert.

Wie schon erwähnt, ist für die drahtlose Kommunikation eine schnelle Verbindung sinnvoll, zumindest beim Laden der Karten. Allerdings könnte nach dem Laden der Karte für ein Gebiet auch auf eine langsame Verbindung gewechselt werden, solange keine großen Datenmengen abgefragt werden.

Die Erreichbarkeit ist aufgrund der hohen Mobilität variabel: bei einem stadtweiten Einsatz kommt es immer dazu, dass sich der Benutzer in einem Funkloch aufhält.

Die Datenverarbeitung ist einfach, da komplexe Aufgaben wie Anfrageverarbeitung, Navigationsberechnung oder Karten in Mehrwertdienste der Nexus-Plattform ausgelagert wurden.

Die Lokalisierung erfolgt auf dem Endgerät über Sensoren oder durch den Benutzer selbst.

Für den *NexusScout* ist das Standard-Schema des Umgebungsmodells ausreichend, auf das er auch nur lesend zugreift.

8.4 *We are different*: ein ortsbezogenes Rollenspiel

We are different [Fre03] ist ein ortsbasiertes Rollenspiel. Das bedeutet, dass der Benutzer (im Folgenden Spieler genannt) sich mit seiner Spielfigur identifiziert und ihre Handlungen und Interaktionen bestimmt, also im Spiel ihre Rolle übernimmt.

Spieler können jederzeit in das Spiel ein- und aussteigen. Das Spiel findet immer und überall statt. Ein Spieler entscheidet jedoch von Fall zu Fall, ob er an einem Spielereignis teilnehmen möchte oder nicht. Das Spielfeld ist die reale Welt, die jedoch mit einer zusätzlichen Bedeutung aus der Spielwelt überlagert wird: so können bestimmte physische Orte eine zusätzliche Spielfunktion bekommen.

Die Geschichte des Spiels unterstützt diese Dualität zwischen Alltagswelt und Spielwelt: sie basiert auf zwei uralten fiktiven Kulturen, deren Mitglieder unerkannt unter uns leben und sich gegenseitig nicht wohlgesonnen sind. Sie verfügen über magische Kräfte, mit denen sie besondere Orte und Gegenstände erkennen können. Jeder Spieler verkörpert eine selbst erdachte Figur (einen sog. Charakter) aus einer der beiden Kulturen. Dieser wird durch eine Beschreibung sowie durch bestimmte Attribute dargestellt, die seine Fähigkeiten im Spiel bestimmen (z.B. körperliche Stärke, Kampffertigkeiten, Gesundheit,...).

Wenn Spieler in der realen Welt aufeinander treffen, so haben sie die Wahl, ob sie im Spielkontext oder auf herkömmliche Weise miteinander agieren. Im Spielkontext können sie nun entweder kämpfen (was durch ein Computerspiel simuliert wird), handeln (mit Spielgeld und/oder Gegenständen aus dem Spiel) oder einfach nur kommunizieren (und dabei die Rolle des Charakter spielen). Weitere Spielereignisse sind das Betreten eines im Spiel relevanten Orts (z.B. einer reale Kirche, die im Spiel mystische Heilung vollbringen kann), oder das Ablegen oder Aufnehmen eines Spielgegenstands (z.B. einer virtuellen Waffe).

Das Spielsystem besteht aus einer Spielanwendung auf einem mobilen Endgerät und einem speziellen Spielserver. Zusätzlich nutzt das Spiel die Nexus-Plattform (siehe Abbildung 33).

Die Spielanwendung bildet die Schnittstelle zum Spieler. Hiermit kann er einen neuen Charakter erstellen, diesen beim Spielserver registrieren, den Zustand seines Charakters einsehen (Gesundheit, Besitz,...), Spielereignisse auslösen und auf Spielereignisse reagieren.

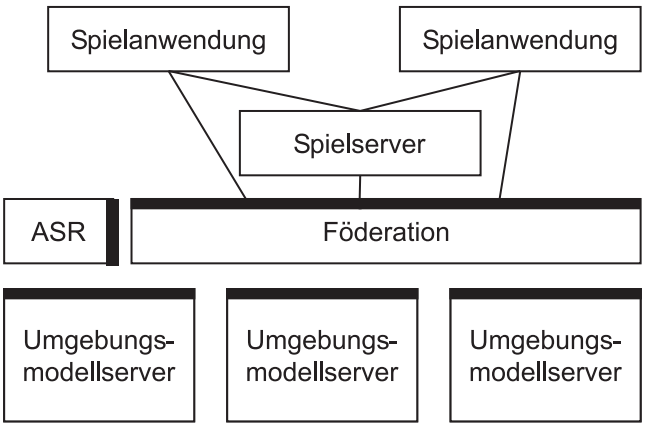

Abbildung 33: Architektur von *We are differrent*

Der Spielserver ist für die Verwaltung der Spielerdaten zuständig. Außerdem stellt er sicher, dass die Spielregeln befolgt werden: Charaktere dürfen nur zugelassene Spielgegenstände haben, und die Spielereignisse müssen gültig sein. Ansonsten wäre es möglich, dass Spieler ihre Spielanwendung verändern, um sich damit unzulässige Vorteile im Spiel zu verschaffen (z.B. zusätzliche, mächtige Gegenstände oder einen unverwundbaren Charakter).

Die Nexus-Plattform schließlich übernimmt die Verwaltung der Spielwelt in Form von Umgebungsmodelldaten, die dann durch Informationen aus dem Spielserver überlagert werden. Betritt z.B. ein Spieler eine Kirche, so kann der Spielserver über das erweiterte Schema des Spiels feststellen, dass dies ein spielrelevanter Ort ist und dem Spieler die dort erlaubten Aktionen ermöglichen (z.B. den Gesundheitswert seines Charakters verbessern).

Tabelle 9 gibt einen Überblick über die Entwurfsentscheidungen, die für *We are different* getroffen wurden.

Die Mobilität soll unbegrenzt sein: selbst wenn für einen abgelegenen Ort keine Spielobjekte definiert wurden, so könnten dort trotzdem Spieler aufeinander treffen und miteinander agieren.

Die Datenspeicherung findet hybrid statt: die reale Welt und auch die Spielobjekte werden in der Nexus-Plattform gespeichert, aufgenommene Gegenstände und die Charaktere auf dem Endgerät (und auf dem Spielserver, um Schummeln zu verhindern).

Tabelle 9: Entwurfsentscheidungen *We are different*

Aspekt	*We are different*
Mobilität	unbegrenzt
Datenspeicherung	hybrid
Drahtlose Kommunikation	langsam (hybrid)
Erreichbarkeit	variabel
Datenverarbeitung	einfach
Lokalisierung	lokal, Benutzer
Umgebungsmodellschema	erweitert
Nutzung des Umgebungsmodells	hybrid

Für die Kommunikation reichen langsame Verbindungen aus, da keine großen Datenmengen übertragen werden. Sollen Spielanwendungen direkt miteinander kommunizieren, so könnten sie zu einem schnellen Ad-Hoc-Modus wechseln (z.B. Bluetooth). Da in diesem Fall der Spielserver die Interaktion nicht mehr kontrollieren kann, müssten jedoch zunächst geeignete Verfahren entwickelt werden, um Schummeln zu verhindern (z.B. Prüfsummen oder Zertifikate).

Die Erreichbarkeit der Endgeräte ist variabel, da die Spieler entscheiden, wann sie am Spiel teilnehmen und wann nicht.

Die Datenverarbeitung ist einfach, es sind für den Spielmechanismus keine komplexen Algorithmen notwendig.

Die Lokalisierung erfolgt lokal über GPS im Außenbereich. Für den Innenraum-Bereich wird zusätzlich die Möglichkeit der Lokalisierung durch den Benutzer gegeben.

Die Idee, für das Spiel die reale Welt mit der Spielwirklichkeit zu überlagern, lässt sich sehr einfach auf das Datenschema abbilden. Für das Spiel wurde ein erweitertes Schema definiert, das im Spiel relevante Objekte von ihren realen Pendants ableitet und diese um die Spieleigenschaften erweitert (z.B. den Charakter von **Person**, die mystische Kirche von **Church**). Zudem gibt es neue Objektklassen für die virtuellen Spielobjekte (Gegenstände, Waffen), die als Kinder von **SpatialObject** modelliert sind.

Die Nutzung des Umgebungsmodells ist hybrid: es werden sowohl Daten gelesen als auch geschrieben (um z.B. Spielobjekte einzufügen).

Zusammenfassend ist zu sagen, dass das Umgebungsmodell und die Nexus-Plattform Spielanwendungen wie *We are different* überhaupt erst ermöglicht. Es ist nicht zu erwarten, dass nur für ein solches Spiel ein großangelegtes räumliches Modell aufgebaut wird. Wenn jedoch ein solches Modell bereits existiert, da es für viele andere Anwendungen entwickelt und verwaltet wird, so ist es durchaus denkbar, dass darüber eine neue Spielbedeutung definiert wird und eine Infrastruktur in Form des Spielservers geschaffen wird, die dann das Spiel selbst ermöglicht. Dieses Vorgehen wird zudem durch die Möglichkeit der erweiterten Schemata stark vereinfacht.

8.5 Die *NexusRallye:* eine explorative Anwendung

Auch die *NexusRallye* [NHM+04] ist ein ortsbezogenes Spiel. Sie ist eine explorative Anwendung, d.h. sie fordert den Benutzer aktiv dazu heraus, seine Umgebung zu erkunden und kennen zu lernen.

Dazu werden Aufgaben in der Umgebung verteilt, die es zu lösen gilt. Es gibt zwei Typen von Aufgaben

- *QuestionTasks* stellen eine Frage und erwarten eine Antwort vom Benutzer (z.B. eine Zahl, eine Auswahl oder einen Freitext). Um dem Ziel des Spiels zu folgen, erfordert dies meist Wissen über die Umgebung, das sich der Benutzer erst durch Erkundung aneignen muss. Je nach Antworttyp können *QuestionTasks* entweder durch das System bewertet werden oder müssen zur Bewertung z.B. an eine Jury gesandt werden.
- *ActionTasks* erfordern als Lösung eine Aktion in der physischen Welt, die von Sensoren beobachtet werden kann. Ein *MoveToTask* könnte lauten: „Gehen Sie zur Mensa“. Die Position des Spielers wird über GPS ermittelt, und die Aufgabe ist gelöst, sobald er sich tatsächlich in unmittelbarer Nähe der Mensa befindet. Denkbar sind auch *BringToTasks*, bei dem bestimmte Gegenstände an einen Ort gebracht werden müssen, an dem sie erfasst werden können (z.B. ein Buch in die Bibliothek, der Sensor wäre in diesem Fall der Barcode-Leser).

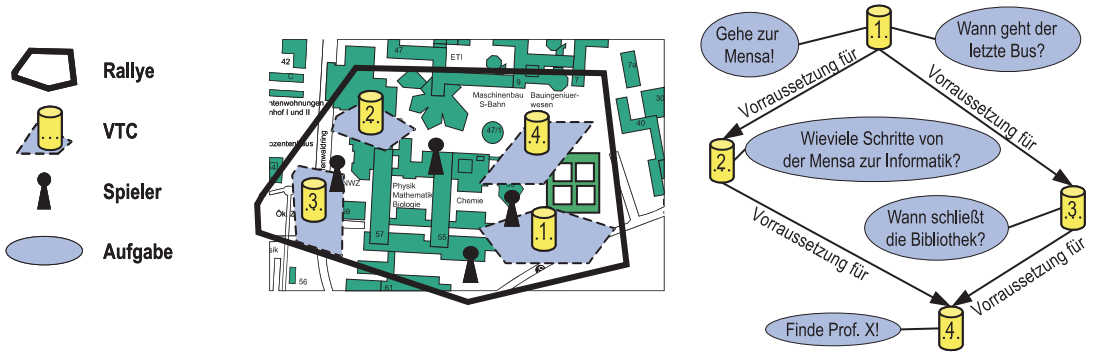

Abbildung 34: Beispiel einer NexusRallye

Ein Entwurfsziel der *NexusRallye* war die Wiederverwendbarkeit der Aufgaben. So ist es möglich, die gleiche Aufgabe in verschiedenen Rallyes einzusetzen (die z.B. für verschiedene Benutzergruppen kreiert wurden). Deswegen sind Aufgaben als eigenständige Objekte modelliert, die keinen eigenen Ortsbezug haben: die Aufgabe „Gehe zur Mensa“ könnte schließlich an vielen Orten sinnvoll gestellt werden (nur nicht an der Mensa selbst).

Für die Gruppierung und Anbindung an konkrete Orte wurde das Konzept der *Virtual Task Container (VTC)* entwickelt, analog zu der Metapher virtueller Litfaßsäulen [LKR99]. Ein VTC hat eine Position, einen Sichtbarkeitsbereich, verweist auf eine Rallye und auf eine oder mehrere Aufgaben. Zudem kann er auf andere VTCs verweisen, die gelöst sein müssen, bevor der Spieler den aktuellen VTC sehen kann (Voraussetzung). Betritt ein Spieler den Sichtbarkeitsbereich eines VTCs, so erhält er diese Aufgaben zur Lösung vorgelegt, sofern er die Voraussetzungen erfüllt.

Abbildung 34 zeigt ein Beispiel für eine solche Rallye: in der Mitte ist der Campus der Universität Stuttgart zu sehen. Die Rallye besteht aus vier VTCs, die jeweils auf eine Aufgabe verweisen. Die Voraussetzungen bilden einen Graph, der auf der rechten Seite abgebildet ist.

Neben Aufgaben und VTCs enthält das Datenmodell einer *NexusRallye* noch **Player** (abgeleitet von **Person**) und **Rallye** (abgeleitet von **SpatialObject**). Der **extent** einer Rallye gibt an, in welchem Gebiet ihre VTCs zu finden sind.

Um die Rallye zu spielen, sucht die Rallyeanwendung zunächst nach Rallye-Objekten, die sich mit der Position ihrer Spielerin überlappen oder in ihrer Nähe liegen. Werden welche gefunden, so kann die Spielerin eine Rallye auswählen und sich zu ihr anmelden. Sie erhält die Startaufgabe der Rallye (meist ein *MoveToTask*, der sie zum ersten VTC führt). Durch die Beantwortung von *QuestionTask* und die Erfüllung weiterer *ActionTasks* kann die Spielerin nun die Aufgaben der Rallye erfüllen und Punkte sammeln. Sind alle Aufgaben gelöst oder eine bestimmte Zeit abgelaufen, endet die Rallye und die Spielerin erhält eine Auswertung.

Analog zum ortsbasierten Rollenspiel *We are different* (Kapitel 8.4, Abbildung 33) kommt auch bei der *NexusRallye* ein spezieller Spielserver zum Einsatz, der für An- und Abmeldungen der Spieler sowie für Auswertungen zuständig ist.

Der Spielablauf wird durch die Verbindung der VTCs und ihrer Aufgaben gegeben. Eine besondere Verantwortung kommt dabei dem Rallye-Designer zu: er muss die VTCs und Aufgaben so definieren, dass die Spieler tatsächlich das Spielziel erreichen können und die Umgebung dabei kennen lernen. Durch die Kombination der Aufgaben werden so mehrere mögliche Pfade durch die Rallye-Umgebung angelegt, denen die Spieler bei der Lösung folgen.

Tabelle 10: Entwurfsentscheidungen *NexusRallye*

Aspekt	***NexusRallye***
Mobilität	(unbegrenzt)
Datenspeicherung	entfernt
Drahtlose Kommunikation	schnell
Erreichbarkeit	variabel
Datenverarbeitung	einfach
Lokalisierung	lokal
Umgebungsmodellschema	erweitert
Nutzung des Umgebungsmodells	lesen

Tabelle 10 zeigt die Entwurfsentscheidungen für die *NexusRallye.*

Prinzipiell kann die Anwendung unbegrenzt eingesetzt werden, sofern am aktuellen Ort der Benutzerin Rallye-, VTC- und Aufgabenobjekte angelegt wurden. Während eines konkreten Spielablaufs ist die Rallye jedoch auf das Rallyegebiet begrenzt.

Die Datenspeicherung erfolgt entfernt, allenfalls die gerade in Bearbeitung befindlichen Aufgaben müssen auf dem Endgerät gehalten werden.

Da die Aufgaben auch Multimedia-Anteile haben können (z.B. ein Bild oder Video, dessen Ursprung man im Rahmen eines *MoveToTask* finden soll), ist eine schnelle drahtlose Kommunikation von Vorteil.

Die Erreichbarkeit ist variabel: wenn sich Spieler in Funklöchern aufhalten, sind sie durch die Plattform nicht erreichbar. Auch dies sollte beim Design einer Rallye berücksichtigt werden: ein VTC sollte nicht in einem Funkloch stehen, da die Spieler dort sonst keine Aufgaben erhalten können.

Die Datenverarbeitung ist einfach, da die Spiellogik bereits über die Verknüpfung der Aufgaben gegeben ist und die Anwendung selbst nur noch Aufgaben laden, anzeigen, und gegebenenfalls mit einer Lösung überprüfen muss.

Die Lokalisierung sollte lokal, aber automatisch erfolgen, damit die *MoveToTasks* überprüft werden können (sonst könnte sich der Benutzer ja einfach per Klick an den Zielort bewegen).

Das Umgebungsmodellschema wurde um VTCs, Rallyes, Spieler und Aufgaben erweitert und wird nur lesend genutzt.

8.6 Fazit

Die Betrachtung verschiedener ortsbasierter Anwendungen zeigt, dass bereits einfache Anwendungen, die auch mit lokaler Datenspeicherung ohne das Umgebungsmodell entwickelt werden können, von der Nexus-Plattform profitieren können (Kapitel 8.2). Sobald erweiterte Anwendungsfälle unterstützt werden sollen, ist eine entfernte und modellübergreifende Plattform notwendig, wie sie in der vorliegenden Arbeit konzipiert wurde (Kapitel 8.3). Durch den Einsatz dieser Lösung werden manche neue Anwendungen erst ermöglicht (Kapitel 8.4), und es konnte mit explorativen Anwendungen sogar ein neues Anwendungsgebiet erschlossen werden (Kapitel 8.5).

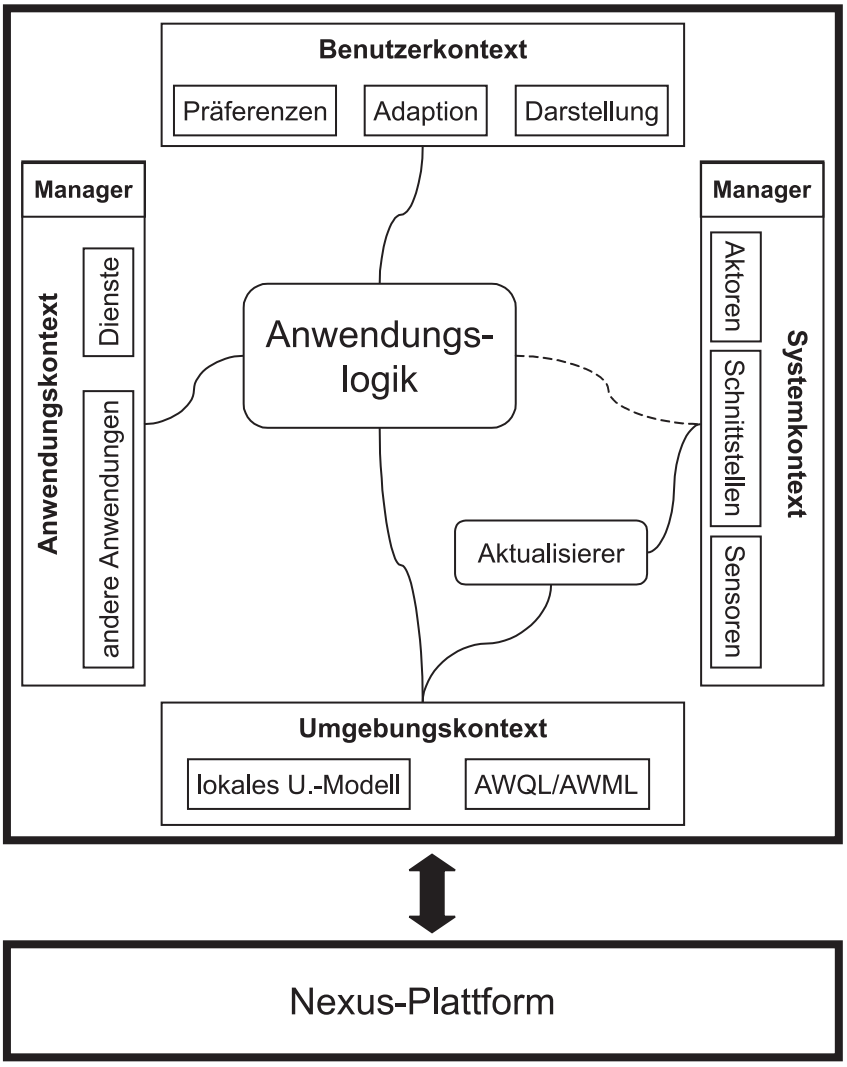

Abbildung 35: Allgemeiner Entwurf einer kontextbezogenen Anwendung

Aus den Erfahrungen mit verschiedenen Anwendungsentwicklungen lässt sich ein allgemeiner Entwurf für kontextbezogene Anwendungen ableiten. Es zeigt sich, dass diese für ihre Adaption im wesentlichen vier verschiedene Kontextbereiche benötigen: den Umgebungskontext, den Systemkontext, den Anwendungskontext und den Benutzerkontext. In dieser Arbeit wurde im wesentlichen der Umgebungskontext betrachtet. Der Umgang mit diesem Kontext lässt sich jedoch verallgemeinern (siehe Abbildung 35) und auf die anderen Kontextbereiche übertragen.

Im Zentrum des Entwurfs steht die Anwendungslogik, die den eigentlichen Zweck der Anwendung implementiert. Sie kann sich vier verschiedener Kontexte bedienen:

- Der Benutzerkontext bestimmt die Schnittstelle zum Benutzer. Abhängig von dessen Präferenzen kann die Darstellung angepasst weden. Hierfür können Adaptionsmechanismen zur Verfügung stehen.
- Über den Anwendungskontext kann die Anwendung erfahren, welche anderen Anwendungen oder Dienste auf dem gleichen oder entfernten Systemen vorhanden sind und wie diese genutzt werden können. Für das Verwalten dieser Dienste ist ein Manager zuständig, der dafür sorgt, dass zur Laufzeit die Dienste neu gebunden werden können.
- Der Systemkontext bestimmt den aktuellen Zustand des Systems, in dem die Anwendung abläuft, und zwar dessen Schnittstellen, Aktoren und Sensoren. Auch hier gibt es einen Manager, der dynamisches Binden und Verwenden von Sensoren, Aktoren und Schnittstellen ermöglicht.
- Der Umgebungskontext enthält schließlich Informationen über die physische Umgebung der Anwendung. Diese können über die Nexus-Plattform angefragt, aktualisiert und auch persistent gemacht werden. Dazu werden Anfragen in AQML formuliert und die Ergebnisse aus AWML heraus deserialisiert und in einem internen Format der Anwendungslogik zur Verfügung gestellt. Bei lokaler oder hybrider Datenspeicherung enthält der Umgebungskontext auch das lokale Umgebungsmodell.

Die Anwendungslogik kann Informationen aus diesen vier Kontexten verwenden.

Ein Aktualisierer ist eine spezielle Anwendung, die den Systemkontext im Umgebungskontext nachführt und z.B. Informationen lokaler Sensoren direkt im lokalen Umgebungsmodell oder in der Nexus-Plattform speichert. In diesem Fall kann eine Anwendung darauf verzichten, direkt mit ihrem lokalen System zu interagieren (gestrichelte Linie). Stattdessen arbeitet sie nur mit ihrem Umgebungskontext, der den Systemkontext enthält (z.B. als technische Objekte, siehe Kapitel 6.7.5). Dadurch wird eine lose Kopplung zwischen der Anwendung und ihrem Systemkontext erreicht, was die Portabilität einer Anwendung auf verschiedene Endgeräte stark erleichtert.

Man muss viel lernen, um zu erkennen, dass man wenig weiß.
(Michel Eyquem Seigneur de Montaigne)

Kapitel 9: Zusammenfassung und Ausblick

In diesem Kapitel wird ein Fazit aus der Arbeit gezogen und ein Ausblick auf zukünftige Erweiterungsmöglichkeiten der vorgestellten Lösung gegeben. Geplante Erweiterungen des Umgebungsmodells wurden bereits in Kapitel 7.3 vorgestellt. Hier geht es um grundlegende Weiterentwicklungen, die sowohl die Modellierung als auch die Modellverwaltung betreffen: die Übertragbarkeit auf andere Anwendungsdomänen, die Modellierung und Berücksichtigung von Inferenzen, Datenqualität, Konsistenz und unscharfen Informationen sowie die strombasierte Datenverarbeitung.

9.1 Fazit

Die vorliegende Arbeit hat sich zum Ziel gesetzt, ortsbezogene Anwendungen durch Integration verschiedener Datenquellen in ein umfassendes Umgebungsmodell zu unterstützen, das von einer offenen Föderationsplattform verwaltet wird. Dazu wurden zunächst allgemeine Anforderungen an das Umgebungsmodell und die Plattform erarbeitet (Kapitel 2). Um diese zu erfüllen, wurde die Anwendungsdomäne ortsbezogener Anwendungen analysiert, um anhand von konkreten Anwendungsfällen inhaltliche Anforderungen an das Umgebungsmodell zu erhalten (Kapitel 4). Da bestehende Architekturen und Plattformen das Anforderungsprofil nicht erfüllen können (Kapitel 3), wurde eine offene Systemplattform konzipiert, die in der Lage ist, ein umfassendes Umgebungsmodell als Föderation lokaler Umgebungsmodelle autonomer Datenanbieter zu verwalten (Kapitel 5).

Das Umgebungsmodell selbst (Kapitel 6) folgt Entwurfskriterien, die sich teilweise aus der Anwendungsdomäne, teilweise aus dem Anforderungsprofil ergeben. Es ist objektbasiert und wird durch ein Standard-Schema,

das durch erweiterte Schemata ergänzt werden kann, spezifiziert. Es kann zwischen statischen, dynamischen und mobilen Objekten unterscheiden sowie zwischen realen (Repräsentationen von in der physischen Welt vorhandenen) und virtuellen Objekten. Durch den Aufbau der Objekt-ID können Objekte sowohl eindeutig identifiziert werden sowie einfach aufgefunden werden. Für den flexiblen Zugriff auf das Umgebungsmodell wurde eine einfache räumliche Anfragesprache entwickelt, die es Anwendungen ermöglicht, heterogene Ergebnismengen zu spezifizieren, die dann in einer Serialisierungssprache zurück geliefert werden. Dies wird durch die semantische Eindeutigkeit von Attributen über Objekttypen hinweg möglich.

Um die Brauchbarkeit der Lösung zu demonstrieren und zu evaluieren, wurden in Kapitel 7 Möglichkeiten dargestellt, wie das Modell in verschiedene Richtungen angepasst, erweitert und weiterentwickelt werden kann. Schließlich wurden Anwendungen entwickelt, die Plattform und Umgebungsmodell nutzen (Kapitel 8) und eine Konzeption für einen allgemeinen Entwurf für mobile, kontextbezogene Anwendungen abgeleitet.

Tabelle 11 zeigt für die einzelnen Anforderungen aus Kapitel 2 auf, wie sie durch diese Arbeit umgesetzt wurden.

Tabelle 11: Anforderungen und ihre Umsetzung

Anforderung	erfüllt durch
Anforderung 1 (Ortsbasierter Zugriff auf Information)	• räumliche Attribute auf einem geometrischen Lokationsmodell für fast alle Objekttypen des Umgebungsmodells (Kapitel 6.2.6, 6.2.7 und 6.7.2) • räumliche Prädikate in der Anfragesprache (Kapitel 6.9.1) • räumliche Verknüpfung von Informationen mit der physischen Welt durch virtuelle Objekte (Kapitel 6.2.11 und 6.7.4)
Anforderung 2 (Ortsbasiertes Ablegen von Information)	• *insert*-Funktionalität der Föderationskomponente und der Umgebungsmodellserver (Kapitel 5.3.2)
Anforderung 3 (ID-basierter Zugriff auf Information)	• *NOL*-Attribut für fast alle Objekttypen des Umgebungsmodells (Kapitel 6.2.2 und 6.7.1)

Tabelle 11: Anforderungen und ihre Umsetzung

Anforderung	erfüllt durch
Anforderung 4 (Integration bestehender Informationsdienste)	• Virtuelle Objekte im Umgebungsmodell (Kapitel 6.2.11 und 6.7.4) • erweiterte Schemata (Kapitel 6.2.12 und Kapitel 7.1) • Integrationsstudien (Kapitel 7.2)
Anforderung 5 (Verlagerung rechenintensiver Operationen in die Plattform)	• Verteilung der Anfrage und Ergebnisintegration durch die Föderationskomponente (Kapitel 5.2.3) • Mehrwertdienste (Kapitel 5.2.4)
Anforderung 6 (Erweiterbarkeit bezüglich neuer Anwendungen)	• Flexible Schnittstelle durch Anfragesprache statt festgelegter Funktionen (Kapitel 6.9) • erweiterte Schemata (Kapitel 6.2.12 und Kapitel 7.1)
Anforderung 7 (Skalierbarkeit)	• Allgemeine Architektur der Plattform: räumliche Organisation der Daten, Umgebungsmodellserver mit eingeschränktem Dienstgebiet, die über einen Verzeichnisdienst den Anfragen zugeordnet werden • Föderationskomponente ohne ‘Gedächtnis’, die einfach repliziert werden kann (Kapitel 5.2) • Objektbasierung des Umgebungsmodells ohne komplexe Relationen-Struktur (Kapitel 6.2.1)
Anforderung 8 (Offenheit)	• bezüglich neuer Datenanbieter: Verzeichnisdienst ermöglicht dynamisches Beitreten und Verlassen der Föderation (Kapitel 5.2.2) • bezüglich neuer Anwendungen: - keine feste Bindung von Anwendungen an die Föderation - Umgebungsmodell als allgemein bekannte semantische Grundlage (im Sinne von Kapitel 3.2) - ansonsten s.o. (Anforderung 6 (Erweiterbarkeit bezüglich neuer Anwendungen)) • bezüglich neuer Dienste: Konzept der Mehrwertdienste (Kapitel 5.2.4) • bezüglich neuer Informationen: erweiterte Schemata (Kapitel 6.2.12 und Kapitel 7.1)

Tabelle 11: Anforderungen und ihre Umsetzung

Anforderung	erfüllt durch
Anforderung 9 (Effizienz)	• räumliche Attribute auf einem geometrischen Lokationsmodell für fast alle Objekttypen des Umgebungsmodells (Kapitel 6.2.6, 6.2.7 und 6.7.2) • dadurch möglich: zweistufiger Index: oberer räumlicher Index durch den Verzeichnisdienst (Kapitel 5.2.2), Umgebungsmodellserver können eigenen räumlichen Index implementieren • Möglichkeit zu Optimierungen durch Caching in der Plattform (Kapitel 5.3.7)
Anforderung 10 (Mächtigkeit des Umgebungsmodells)	• Analyse der Anwendungsdomäne und konkreter Szenarien (Kapitel 4) • erweiterte Schemata (Kapitel 6.2.12 und Kapitel 7.1) • Entwicklung von Beispielanwendungen (Kapitel 8)
Anforderung 11 (Effiziente Verwaltung des Umgebungsmodells)	• Objektbasierung des Umgebungsmodells ohne komplexe Relationen-Struktur (Kapitel 6.2.1) • Anfragesprache ohne komplexe Operatoren wie z.B. *Joins* (Kapitel 6.9)
Anforderung 12 (Erweiterbarkeit des Umgebungsmodells)	• erweiterte Schemata (Kapitel 6.2.12 und Kapitel 7.1)
Anforderung 13 (Einfachheit des Umgebungsmodells)	• allgemeiner Entwurf des Umgebungsmodells (Kapitel 6.2); z.B.: - Position *und* Ausdehnung für alle räumlichen Objekte (Kapitel 6.7.1) - unidirektionale Relationen (Kapitel 6.2.9) • 'menschenlesbare' Serialisierungssprache (Kapitel 6.8)

Zusammenfassend ist zu sagen, dass die Anforderungen aus Kapitel 2 weitestgehend umgesetzt wurden. Handlungsbedarf besteht vor allem noch bei der konkreten Umsetzung der einzelnen Plattform-Komponenten (insbesondere Föderation und Umgebungsmodellserver), um die Erfüllung der Anforderung 9 (Effizienz) auch durch Messungen an einer Implementierung belegen zu können. Hierzu konnte in dieser Arbeit nur die Grundlage gelegt werden.

Wie bereits zu Beginn der Arbeit erwähnt wurde, wird das Modell und die Plattform im Nexus-Projekt eingesetzt und dort beständig weiterentwickelt, was auch zu tiefergehenden Lösungen zu den allgemeinen Anforderungen führt. Im Folgenden wird ein kurzer Ausblick auf diese Arbeiten und Vorhaben gegeben. Die Arbeit schließt mit einer Betrachtung, wie die Ansätze auf andere Anwendungsdomänen übertragen werden können.

9.2 Inferenzen und Regeln

In Kapitel 3.2.6 wurde das Umgebungsmodell gegenüber ontologiebasierten Ansätzen dadurch abgrenzt, dass es keine Regeln und Inferenzen abbildet, mit denen aus bestehenden Umgebungsinformationen neues Wissen abgeleitet werden kann. Der Grund hierfür liegt in der Skalierbarkeit: komplexe Inferenzsysteme werden auf großen Datenmengen schnell sehr ineffizient. Die vierte Ebene räumlicher Ontologien nach Andrew Frank (Kapitel 3.2.5) wird in der Nexus-Plattform den Anwendungen überlassen.

Trotzdem wäre es wünschenswert, semantische Regeln auf Umgebungsinformation definieren und auch in der Plattform verwalten zu können: viele Regeln kontextbezogener Anwendungen beziehen sich auf die Ableitung von „höherer“ Kontextinformation aus „niederer“ Kontextinformation, z.B. Benutzeraktivitäten („bei der Arbeit“) aus verschiedenen Sensormessungen und Kalendereinträgen. Wenn solche Regeln und deren abgeleitetes Wissen von der Plattform verwaltet werden würde, könnten Anwendungen es teilen und so besser unterstützt werden.

In [BN04] wurde ein kombinierter Ansatz vorgestellt, mit dem die Skalierbarkeit des Umgebungsmodells und der Nexus-Plattform mit der Mächtigkeit ontologiebasierter Arbeiten verbunden werden könnte. Die Idee ist, zunächst den Teil des gesamten Modells zu selektieren, der für die Anwendung einer bestimmten Regel (oder eine Regelmenge) notwendig ist. Ein Inferenzdienst könnte dann auf dieser beschränkten Menge von Objekten effizient arbeiten und das Ergebnis verschiedenen Anwendungen zur Verfügung stellen.

In Abbildung 36 ist graphisch dargestellt, wie sich die Nexus-Plattform auf die Ontologie-Ebenen abbildet: Die physische Welt wird von Sensoren beobachtet (Ebene 1). Sie aktualisieren das Umgebungsmodell, das

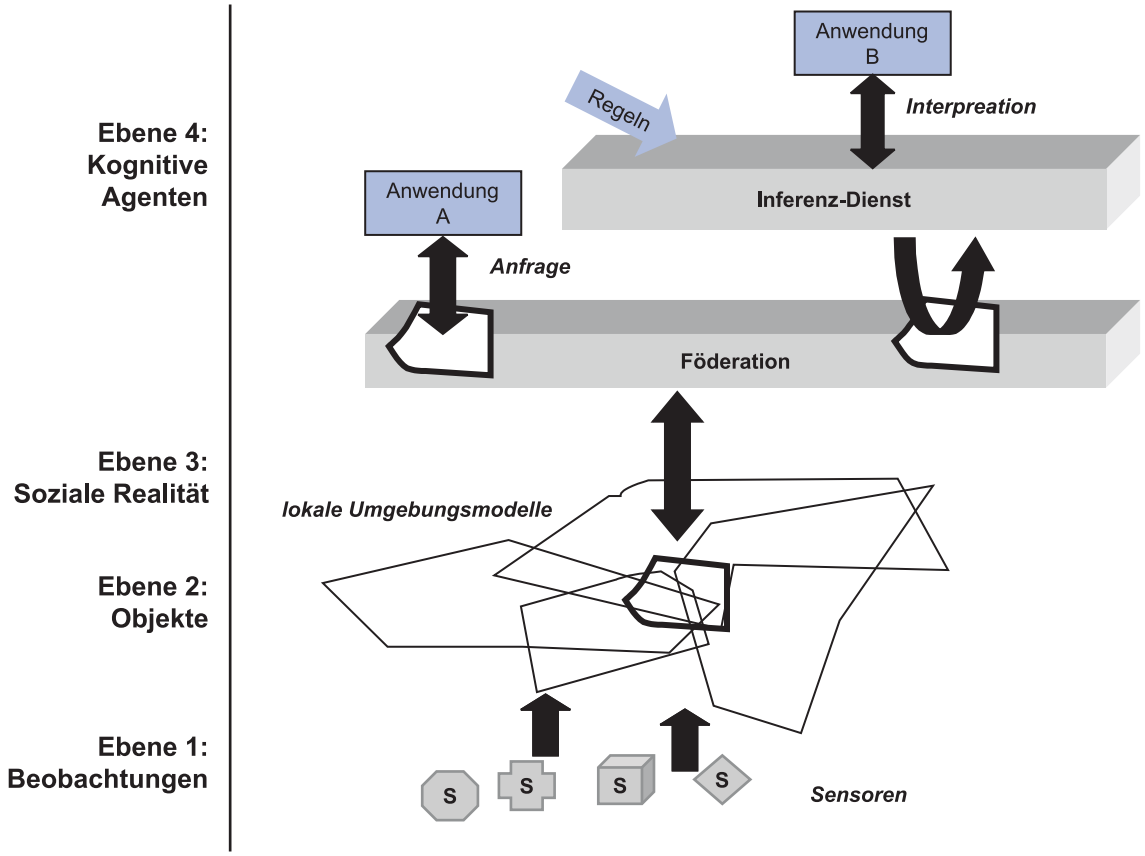

Abbildung 36: Erweiterung der Plattform um einen Inferenzdienst (nach [BN04])

die Beobachtungen zu Objekten gruppiert (Ebene 2) und auch Vereinbarungen der sozialen Realität abbildet (Ebene 3). Bisherige Nexus-Anwendungen (Anwendung A) greifen über die herkömmliche Anfrageschnittstelle auf die Föderation zu und müssen ihre Schlussfolgerungen aus den angefragten Daten selbst treffen. Damit gehören sie zur Ebene 4 der kognitiven Agenten. Ein wie oben skizzierter Inferenz-Dienst könnte ebenfalls die Anfrageschnittstelle verwenden, um einen Weltausschnitt auszuwählen. Auf diesem könnte er mit Hilfe von Regeln ähnliche Schlussfolgerungen vornehmen und deren Ergebnisse neuen Anwendungen (Anwendung B) zur Verfügung stellen. Eine offene Frage hierbei ist, wie die notwendigen Regeln definiert werden und in das System kommen. Sie könnten von Anwendungen dem Inferenzdienst zur Verfügung gestellt werden oder zusammen mit den Daten im Umgebungsmodell gespeichert werden. In diesem Fall müsste das Umgebungsmodell erweitert und eine Konzeption für Regeln mit und ohne Ortsbezug entwickelt werden.

9.3 Strombasierte Verarbeitung

Wie bereits in Kapitel 7.3.3 angedeutet, benötigen manche kontextbezogene Anwendungen Datentypen, die am sinnvollsten in einem Datenstrom übertragen werden, z.B. Audio- oder Videodaten. Auch kontinuierliche Sensormessungen können als Datenstrom aufgefasst werden. In der bisherigen Plattform ist dies nicht vorgesehen, da auf eine Anfrage hin das Ergebnis in einem einzelnen Dokument komplett übertragen wird. Eine Erweiterung hin zu strombasierter Verarbeitung ist zudem sinnvoll um kontinuierliche Anfragen unterstützen zu können, bei denen sich das Anfragegebiet mit der Bewegung des anfragenden Benutzers ständig ändert.

Mögliche Konsequenzen einer solchen Erweiterung sind:

- Die Kommunikationsverbindung zwischen Plattform und Anwendung muss über einen längeren Zeitraum hin aufrecht erhalten werden. Dies erfordert ein Sessionkonzept.
- Je nach Anforderungen des Datenstroms muss dabei eine gewisse Übertragungs-Qualität (*quality of service*) sichergestellt werden.
- Das Format der Ergebnisdokumente muss unter Umständen angepasst werden: statt einer unsortieren Objektmenge ist eine nach Distanz sortierte Objektmenge oder auch nur einzelne Attribute von Objekten sinnvoll.
- Um weiterhin skalierbar zu bleiben, könnte in der Plattform das Zusammenführen ähnlicher Datenströme notwendig werden. Hierfür könnten Ansätze bestehender strombasierter Arbeiten verwendet werden, z.B. *NiagaraCQ* [CDT+00] oder *Place* [MAH+03]: hier werden die Anfragebäume mehrerer kontinuierlicher Anfragen optimiert, um gemeinsame Teilbäume zu vereinigen, räumliche Beziehungen (z.B. *enthalten-in*) auszunutzen oder ähnliche Objektselektionen zu teilen.

9.4 Unschärfe und Konsistenz

Ein Punkt, der in dieser Arbeit lediglich am Rande beachtet wurde, ist die Tatsache, dass ortsbezogene Daten niemals exakt mit der Realität übereinstimmen, sondern unscharf oder im schlimmsten Fall falsch sein können. Dadurch kommt es zu Inkonsistenzen zwischen Modell und Welt und auch zwischen lokalen Umgebungsmodellen, welche die Föderation möglichst vor der Anwendung verbergen sollte, sofern dies möglich ist.

Für die Entstehung solcher Inkonsistenzen gibt es zwei Hauptfaktoren: erstens Fehler bzw. Unschärfe bei der Datenerhebung, und zweitens die Drift, wenn die erhobenen Daten veralten, während sich die Welt verändert.

Um eine Inkonsistenz von Modelldaten überhaupt feststellen zu können, bedarf es eines Vergleichspunkts. Dazu können die Daten mit der Welt selbst (über Sensorbeobachtungen) verglichen werden. Der Ansatz dieser Arbeit, verschiedene lokale Umgebungsmodelle zu föderieren, ermöglicht jedoch auch den Vergleich zu anderen Modellen. Durch die räumliche und thematische Überlappung verschiedener Modelle können Inkonsistenzen festgestellt werden und mit Hilfe geeigneter Verfahren eventuell sogar aufgelöst werden, um die Datenqualität zu verbessern.

Solche Verfahren können jedoch teuer sein: sie kosten z.B. Energie (relevant bei ressourcenbeschränkten Endgeräten in hybriden Netzstrukturen), Zeit (relevant bei zeitkritischen Prozessen, z.B. bei Datenströmen), Geld (relevant in verschiedenen Betreibermodellen) oder beschränkte Ressourcen (z.B. Kommunikationsbandbreite).

Ein Ziel könnten Verfahren sein, die für Anfragen Kosten und Nutzen analysieren und feststellen, ob sich eine Verbesserung der Datenqualität durch zusätzliche Modell- oder Sensorvergleiche lohnt.

Um dies zu erreichen, müssen zunächst Metriken entwickelt werden, um Inkonsistenzen in Umgebungsmodellen zu quantifizieren. In einem zweiten Schritt können dann Verfahren entworfen werden, die verschiedene Modelle miteinander vergleichen und eine quantifizierte zusammengefügte Weltsicht im föderierten Modell ergeben.

Solche Verfahren können an verschiedenen Stellen zur Verbesserung der Datenqualität eingesetzt werden:

- zur Implementierung energieschonender Algorithmen für Datendissimination in hybriden Netzstrukturen,
- für effizientes, echtzeitfähiges strombasiertes Verteilen von Modelldaten,
- für die kostenbewusste Auswahl von Modelldaten verschiedener Betreibern durch die Föderation,
- für die Bereitstellung einer garantierten Datenqualität für sicherheitskritische Anwendungen (z.B. Blindennavigation: „lieber keine Daten als falsche Daten“).

Bei der Einführung solcher Verfahren müssten folgende Punkte beachtet werden:

- Unsicherheit, Ungenauigkeiten und Datenqualität müssten in den Metadaten dargestellt werden können.
- Die Anfragesprache braucht eine erweiterte Semantik. Wenn man nicht mehr von einer exakten Positions eines Objekts ausgeht, ist es u.U. bei räumlichen Prädikaten wünschenswert, eine Wahrscheinlichkeit anzugeben, mit der das Prädikat zutritt. Hierzu könnten die Ansätze aus [Leo03] und [LR02] verwendet werden.

9.5 Übertragbarkeit auf andere Anwendungsdomänen

Lässt sich das Vorgehen und die Lösung dieser Arbeit auch auf andere Anwendungsdomänen übertragen? Dies hängt davon ab, in wie weit die Spezifika der Domäne ortsbezogener Anwendungen in den anderen Domänen wieder zu finden sind.

Ein umfassendes Umgebungsmodell als Integrationsstrategie lässt sich auch für andere Domänen definieren, sofern man diese auf eine Gruppe von Szenarien beschränken kann und für diese dann ein gemeinsames Datenmodell definiert. Für eine Integration, wie sie hier vorgeschlagen wird, ist es notwendig, ein oder mehrere globale Index-Attribute zu finden, wie sie der zweidimensionale Raum für das Umgebungsmodell darstellt. Die Eigenschaft dieses Attributs ist es, über einen Verzeichnisdienst die relevanten Modellserver zur Beantwortung einer Frage heraus zu finden.

Das Modell könnte dann z.B. durch ein Standardisierungs-Gremium definiert werden. Je älter die Anwendungsdomäne jedoch ist, desto mehr existierende Systeme wird es geben, und desto schwieriger wird es werden sich auf gemeinsame Modellierungsstandards zu einigen. Der Ansatz des umfassenden Umgebungsmodells eignet sich daher vor allem für neue Anwendungsdomänen.

Habe nun, ach! Philosophie, Juristerei und Medizin, und leider auch Theologie! durchaus studiert, mit heißem Bemühn. Da steh ich nun, ich armer Tor! Und bin so klug als wie zuvor. (Goethe, Faust I)

Kapitel 10: Literaturverzeichnis

[Ari93] Aristoteles: **Meteora, 2. Buch.** 293 a 15 – 298 a 20

[Att01] Attunity: **Attunity Connect Architecture.** Product white paper, March 2001
http://www.attunity.com/files/AttunityConnectTechWhitePaper.pdf

[AWS] D. Nicklas et al. **Schemadefinitionen des Augmented World Schema.** Online verfügbar, Stand 2005
http://nexus.informatik.uni-stuttgart.de/en/research/documents

[BCM+03] F. Bader, D. Calvanese, D. McGuinness, D. Nardi und P. Patel-Schneider (Hrsg): **The Description Logic Handbook—Theory, Implementation and Applications.** Cambridge University Press, 2003

[BLN86] C. Batini, M. Lenzerini und S.B. Navathe: **A Comparative Analysis of Methodologies for Database Schema Integration.** ACM Computing Surveys, Vol. 18, No. 4, Dec. 1986

[BBH+03] M. Bauer, C. Becker, J. Hähner und G. Schiele: **ContextCube—Providing Context Information Ubiquitously.** Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW 2003), 2003

[BDG+04] M. Bauer, F. Dürr, Jan Geiger, M. Grossmann, N. Hönle, J. Joswig, D. Nicklas und T. Schwarz: **Information Management and Exchange in the Nexus Platform.** Universität Stuttgart: Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Fakultätsbericht Nr. 04, 2004

[BR04] M. Bauer und K. Rothermel: **How to Observe Real-World Events through a Distributed World Model.** Proceedings of the Tenth International Conference on Parallel and Distributed Systems 2004 (ICPADS 2004); Newport Beach, California, July 7-9, 2004

[BKW02] J. Baus, A. Krüger und W. Wahlster: **A resource-adaptive mobile navigation system.** Proceedings of International Conference on Intelligent User Interfaces, San Francisco, 2002

[BN04] C. Becker und D. Nicklas: **Where do spatial context-models end and where do ontologies start? A proposal of a combined approach.** J. Indulska, D. De Roure (Hrsg): Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management in conjunction with UbiComp 2004

[BF00] T. Berners-Lee und M. Fischetti: **Weaving the web: the original design and ultimative destiny of the World Wide Web by its inventors.** San Francisco: HarperCollins, 2000

[BHL01] T. Berners-Lee, J. Hendler und O. Lassila: **The Semantic Web.** Scientific American, May 2001

[BBD+03] F. Bellotti, R. Berta, A. De Gloria, E. Ferretti und M. Margarone: **Designing Mobile Games for a Challenging Experience of the Urban Heritage.** Proceedings of Euro-Par 2003 Parallel Processing, LNCS 2790, Springer, 2003

[Ber00] T. Berners-Lee mit M. Fischetti: **Weaving the Web—The Original Design and Ultimate Destiny of the World Wide Web.** HarperCollins Publishers Inc. New York, 2000

[BFH+01] S. Bjork, J. Falk, R. Hansson und P. Ljungstrand: **Pirates! Using the Physical World as a Game Board.** Proceedings Interact 2001, IFIP, 2001

[BRJ99] G. Booch, J. Rumbaugh und I. Jacobson: **The Uniform Modeling Language User Guide.** Addison Wesley Longman, 1999

[BMR04] S. Bürklen, P. José Marrón und K. Rothermel: **An Enhanced Hoarding Approach Based on Graph Analysis.** Proceedings of the 5th IEEE International Conference on Mobile Data Management (MDM 2004); Berkeley, California, USA; January 19-22, 2004

[Cellpoint] **CellPoint Inc Webseite.** Stand 2002
`http://www.cellpoint.de`

[CDT+00] J. Chen, D. J. DeWitt, F. Tian und Y. Wang: **NiagaraCQ: A Scalable Continous Query System for Internet Databases.** SIGMOD 2000, Proceedings ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 2000

[CFJ04] H. Chen, T. Finin und A. Joshi: **An Ontology for Context-Aware Pervasive Computing Environments.** Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, Cambridge University Press, May 31, 2004

[CDM+00] K. Cheverst, N. Davies, K. Mitchell, A. Friday und C. Efstratiou: **Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences.** Proceedings of CHI 2000, Netherlands 2000

[CSS] H. Wium Lie und B. Bos: **Cascading Style Sheets, level 1.** W3C Recommendation 17 Dec 1996
http://www.w3.org/TR/REC-CSS1-961217.html

[CRF00] D. D. Chamberlin, J. Robie und D. Florescu: **Quilt: An XML Query Language for Heterogeneous Data Sources.** WebDB (Informal Proceedings), Seiten 53-62, 2000

[CK00] G. Chen und D. Kotz: **A Survey of Context-Aware Mobile Computing Research.** Dartmouth Computer Science Technical Report TR2000-381, Dartmouth College (2000)

[Cla03] K. G. Clark: **The Semantic Web is Closer Than You Think.** Published on XML.com
http://www.xml.com/pub/a/2003/08/20/deviant.html

[CLC+02] N. H. Cohen, H. Lei, P. Castro, J. S. Davis II und A. Purakayastha: **Composing Pervasive Data using iQL.** Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02), 2002

[CPW+02] N. H. Cohen, A. Purakayastha, L. Wong und D. L. Yeh: **iQueue: a pervasive data-composition framework.** 3rd International Conference on Mobile Data Management, 2002

[CYC] CyCorp: **CYC Upper Ontology.** Stand 2005
http://www.cyc.com/cycdoc/vocab/vocab-toc.html

[DA99] A. Dey und G. Abowd: **Towards a better understanding of context and context-awareness.** Georgria Tech GVU Technical Report, GIT-GVU-99-22, 1999

[DDG+04] T. Drosdol, F. Dürr, M. Großmann, N. Hönle und S. Volz: **Technischer Bericht der Taskforce Topologie.** Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Sonderforschungsbereich 627 Nexus, interner technischer Bericht, Stand 6.7.2004

[DKK+04] D. Dudkowski, U. Käppeler, G. Kindermann und D. Klinec: **Modellierung von Sensoren und Sensordaten in der Nexus-Plattform.** Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Sonderforschungsbereich 627 Nexus, interner technischer Bericht, Stand 28.9.2004

[DOM02] Projekt-Webseite: **DOM – der orientierte Mensch.** Gefördert durch BMBF, Webseite, Stand 03.05.2002
http://www.der-orientierte-mensch.de

[DR03] F. Dürr und K. Rothermel: **On a Location Model for Fine-Grained Geocast.** A. Dey, A. Schmidt, J. F. McCarthy (Hrsg): Proceedings of the Fifth International Conference on Ubiquitous Computing (UbiComp) ; Seattle, WA, Oct. 12-15, 2003

[Esri] Esri: **ESRI Webseite.** Stand 2002
http://www.esri.com

[EPC] EPCGlobal Inc: **Firmen-Homepage.** Stand 2005
http://www.epcglobalinc.org

[EPSG] European Petroleum Survey Group (EPSG) : **Geodesy Parameters V 6.3.** Stand 2005
http://www.epsg.org

[FTP] J. Postel und J. Reynolds: **FILE TRANSFER PROTOCOL (FTP).** Network Working Group, Request for Comments: 959, ISI, October 1995
http://www.w3.org/Protocols/rfc959

[FAB+03] M. Flintham, R. Anastasi, S. Benford, T. Hemmings, A. Crabtree, C. Greenhalgh, T. Rodden, N. Tandavanitj, M. Adams und J. Row-Farr: **Where on-line meets on-the-streets: experiences with mobile mixed reality games.** Proceedings CHI 2003, Fort Lauderdale, Florida, 5- 10 April 2003

[FAO+03] C. Floerkemeier, D. Anarkat, T. Osinski und M. Harrison: **PML Core Specification 1.0.** Auto-ID Center Recommendation 15 September 2003
http://develop.autoidcenter.org

[FK99] I. Foster und C. Kesselmann (Hrsg.): **The Grid: Blueprint for a Future Computing Infrastructure.** Morgan Kaufman, 1999

[Fra03] A. U. Frank: **Ontology for Spatio-Temporal Databases.** M. Koubarakis et al. (Hrsg), Spatio-Temporal Databases—The CHOROCHRONOS Approach. Lecture Notes in Computer Science, Springer 2003

[Fre03] T. Frei: **Anwendungsentwicklung mit der Nexus-Plattform am Beispiel eines ortsbasierten, mobilen Spiels.** Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Diplomarbeit Nr. 2008, 2003

[Gate5] Gate 5 AG: **Gate 5 AG Webseite.** Stand 2002
http://www.gate5.de

[GPQ97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos und J. Widom: **The TSIMMIS Approach to Mediation: Data Models and Languages.** Journal of Intelligent Information Systems 8(2): Seiten 117-132, 1997

[GSS+02] D. Garlan, D. Siewiorek, A. Smailagic und P. Steenkiste: **Project Aura: Towards Distraction-Free Pervasive Computing.** IEEE Pervasive Computing, special issue on Integrated Pervasive Computing Environments, Vol.1, Nr. 2, 2002

[GJP95] A. Gomez-Perez, N. Juristo und J. Pazos: **Evaluation and asessment of knowledge sharing technology.** N.J. Mars (Hrsg) Towards Very Large Knowledge Bases - Knowledge Building and Knowledge Sharing 1995, Seiten 289-296, IOS Press, Amsterdam, 1995

[GBH+05] M. Grossmann, M. Bauer, N. Hönle, U. Käppeler und D. Nicklas: **Efficiently Managing Context Information for Large-scale Scenarios.** Proceedings of the 3rd IEEE Conference on Pervasive Computing and Communications (PerCom), Kauai Island, Hawaii, March 8-12, 2005

[GML] S. Cox, A. Cuthbert, R. Lake und R. Martell (Hrsg): **Geographic MarkUp Language (GML 2.0).** OpenGIS® Implementation Specification, OGC Document Number: 01-029,20 February 2001 `http://www.opengis.net/gml/01-029/GML2.html`

[Gru93] T. R. Gruber: **A Translation Approach to Portable Ontology Specifications.** Knowledge Acquisition, Volume 5, Special issue: Current issues in knowledge modeling, Academic Press Ltd. London UK, UK, 1993

[Grz00] S. Grzan: **Konzeption und Entwicklung einer mobilen Datenbankanwendung.** Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1794, 2000

[Grz02] S. Grzan: **Enabling technology for an indoor location aware information system.** Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1958, 2002

[GLM02] A. Gupta, B. Ludäscher und M. Martone: **Registering Scientific Information Sources for Semantic Mediation.** Proceedings of the 21st International Conference on Conceptual Modeling, Tampere, Finland, 2002

[HMN+99] L.M. Haas, R.J. Miller, B. Niswonger, M. Tork Roth, P.M. Schwarz und E.L. Wimmers: **Transforming Heterogeneous Data with Database Middleware: Beyond Integration.** IEEE Data Engineering Bulletin, Vol. 22, No. 1, Seiten 31-36, 1999

[Hal01] T. A. Halpin: **Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design.** Morgan Kaufman, San Francisco, 2001

[Hau04] C. Hauser: **Mobility Management Meets Privacy—the Failure of Existing Proposals and a New, Future-Proof Approach.** Proceedings of the Second International Workshop on Mobility Management & Wireless Access Protocols

[HMR02] T. Häusser, M. Mirochnitchenko und M. Rindermann: **Comparison of platforms for Location Based Services.** Fachstudie Nr. 19, Fakultät Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, 2002

[HK03] N. Haala und M. Kada: **Generation and application of virtual landscape models for location-based services.** Proceedings of Geo Sensor Networks 2003

[HBM02] J. Hendler, T. Berners-Lee und E. Miller: **Integrating Applications on the Semantic Web.** Journal of the Institute of Electrical Engineers of Japan, Vol 122(10), Seiten 676-680, October, 2002

[HI04] K. Henricksen und J. Indulska: **A Software Engineering Framework for Context-Aware Pervasive Computing.** Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom), 2004

[HB01] J. Hightower und G. Boriello: **Location Systems for Ubiquitous Computing.** in IEEE Computer 34, 8, Special Issue on Location Aware Computing, Seiten 57-66, 2001

[HKL+99] F. Hohl, Uwe Kubach, A. Leonhardi, K. Rothermel und M. Schwehm: **Next Century Challenges: Nexus—An Open Global Infrastructure for Spatial-Aware Applications.** Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom '99) Seattle, WA, USA, August 1999

[HS04] D. Holberg und O. Schneider: **GEIST – Geschichte vor Ort erleben.** G. Stanke Hrsg.) u.a.; Konferenzband EVA 2004 Berlin. Proceedings: Elektronische Bildverarbeitung & Kunst, Kultur, Historie. Berlin Gesellschaft zur Förderung angewandter Informatik e.V., 2004

[HKN+05] N. Hönle, U. Käppeler, D. Nicklas und T. Schwarz: **Benefits Of Integrating Meta Data Into A Context Model.** 2nd Workshop on Context Modeling and Reasoning (CoMoRea), Workshop-Proceedings of the 3rd IEEE Conference on Pervasice Computing and Communications, PerCom2005, 2005

[HPH03] I. Horrocks, P. F. Patel-Schneider und F. van Harmelen: **From SHIQ and RDF to OWL: The Making of a Web Ontology Language.** Journal of Web Semantics, 2003

[HFT+99] T. Höllerer, S. Feiner, T. Terauchi, G. Rashid und D. Hallaway: **Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system.** Computer and Graphics, 23, 779-785, 1999

[HHT+01] T. Höllerer, D. Hallaway, N. Tienna und S. Feiner: **Steps towards accommodating variable position tracking accuracy in a mobile augmented reality system.** IJCAI-2001 Working Notes: AI in Mobile Systems (AIMS), 31-37, 2001

[HJK+01] W. Hoschek, J. Jaen-Martinez, P. Kunszt, B. Segal, H. Stockinger, K. Stockinger und B. Tierney: **Data Management Architecture Report. Design, Requirements and Evaluation Criteria.** Technical report, DataGrid-02-D2.2, 2001

[HTML] T. Berners-Lee und D. Connolly: **Hypertext Markup Language – 2.** Network Working Group, Request for Comments: 1866, 1995 `http://www.ietf.org/rfc/rfc1866.txt`

[HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach und T. Berners-Lee: **Hypertext Transfer Protocol -- HTTP/1.1.** Network Working Group, Request for Comments: 2616, 1999

[IBM00] IBM Corporation: **IBM DB2 Data Joiner.** Fact sheet, 2000 `http://www-3.ibm.com/software/data/datajoiner/brochure/factsheet.pdf`

[JGN+05] M. Jakob, M. Grossmann, N. Hönle und D. Nicklas: **DCbot: Exploring the Web as Value-added Service for Location-based Applications.** Proceedings of the 21st International Conference on Data Engineering (ICDE) April 5-8, 2005, Tokyo, Japan. IEEE Computer Society, 2005.

[JU99] R. Jasper und M. Uschold: **A Framework for Understanding and Classifying Ontology Applications.** 12th Workshop on Knowledge Acquisition Modeling and Management KAW'99. Published on-line 1999 `http://sern.ucalgary.ca/KSI/KAW/KAW99`

[JFP+01] C. S. Jensen, A. Friis-Christensen, T. B. Pedersen, D. Pfoser, S. Saltenis und N. Tryfona: **Location-Based Services – A Database Perspective.** Proceedings of the Eighth Scandinavian Research Conference on Geographical Information Science, A*s, Norway, June 25-27, 2001

[JS03] G. Judd und P. Steenkiste: **Providing Contextual Information to Pervasive Computing Applications.** in Proceedings of IEEE International Conference on Pervasive Computing and Communications (PerCom), 2003

[KOA+99] C. Kidd, R. Orr, G. Abowd, C. Atkeson, I. Essa, B. B. MacIntyre, E. Mynatt, T. Starner, W. Newstetter: **The Aware Home: A Living Laboratory for Ubiquitous Computing Research.** Proceedings of the Second International Workshop on Cooperative Buildings (CoBuild), 1999

[KLW95] M. Kifer, G. Lausen und J. Wu: **Logical foundations of object-oriented and frame-based languages.** Journal of the ACM, 1995

[KRC+03] S. G. M. Koo, C. Rosenberg, H. Chan, Y. Chung Lee: **Location-Based E-Campus Web Services: From Design to Deployment.** Proceedings of the 1st IEEE Conference on Pervasive Computing and Communications (PerCom) 2003

[Leh03] O. Lehmann: **A Spatial Modelling Infrastructure for Ubicomp Applications in Home Environments.** Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Diplomarbeit Nr. 2075, 2003

[LBB+04] O. Lehmann, M. Bauer, C. Becker und D. Nicklas: **From Home to World—Supporting Context-aware Applications through World Models.** Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom), 2004

[LG90] D.B. Lenat und R.B. Guha: **Building large knowledge-based systems. Represenation and inference in the Cyc project.** Addison-Wesley, Reading, Massachusetts, 1990

[Leo03] A. Leonhardi: **Architektur eines verteilten skalierbaren Lokationsdienstes.** Universität Stuttgart: Dissertation 2003

[LR02] A. Leonhardi und K. Rothermel: **Architecture of a Large scale Location Service.** Proceedings of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS 2002), Vienna, Austria, Seiten 465-466, 2002

[LKR99] A. Leonhardi, U. Kubach, K. Rothermel und A. Fritz: **Virtual Information Towers —A Metaphor for Intuitive, Location-Aware Information Access in a Mobile Environment.** Proceedings of the Third International Symposium on Wearable Computers (ISCW), 1999

[LRO96] A.Y. Levy, A. Rajaraman und J.J. Ordille: **Querying Heterogeneous Informaton Sources Using Source Descriptions.** Proceedings 22nd Conf. on Very Large Databases, Mumbai (Bombay), India, 1996

[LKA+96] S. Long, R. Kooper, G.D. Abowd und C.G. Atkeson: **Rapid prototyping of mobile context-awawre applications: The cyberguide case study.** Proceedings of the Second Annual International Conference on Mobile Computing and Networking (MobiCom), Rye, New York, USA, ACM Press, 1996

[LG04] S. Lück und A. Gutscher: **Improving Access Discovery by Analysing World-Model Information.** 1. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste, 2004

[Mac91] R. MacGregor: **Inside the LOOM classifier.** SIGART Bulletin, 2(3), Juni 1991

[MSS+02] A. Maedche, S. Staab, R. Studer, Y. Sure und R. Volz: **SEAL—Tying up Information Integration and Web Site Management by Ontologies.** Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, Vol. 25, Nr. 1, March 2002

[MZ00] R. Malaka und A. Zipf: **Deep Map – challenging IT research in the framework of a tourist information system.** D.R. Fesenmaier, S. Klein und D. Buhalis (Hrsg): Information and communication technologies in tourism 2000, Springer-Verlag Wien, New York, 2000

[MS00] N. Marmasse und C. Schmandt: **Location-aware information delivery with commotion.** Proceedings of the second International Symposium on Handheld and Ubiquitous Computing (HUC), Bristol, UK, 2000

[Mes01] J. Messmer: **Modellierung der Augmented World in Nexus.** Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1870, 2001

[MoinMoin] **MoinMoin Wiki.** Wiki zur Implementierung: Stand Januar 2005 `http://moinmoin.wikiwikiweb.de/FrontPage`

[MAH+03] M. F. Mokbel, W. G. Aref, S. E. Hambrusch und S. Prabhakar: **Towards scalable location-aware services: requirements and research issues.** Proceedings of the 11th ACM international symposium on Advances in geographic information systems, New Orleans, Louisiana, USA, 2003

[Mot03] S. Motzer: **Query-basierte Abfrage des Lokationsdienstes in Nexus.** Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Studienarbeit Nr. 1884, 2003

[NavTeq] NavTeq: **Firmen-Homepage.** Stand 2005 `http://www.navteq.com`

[Nexus] Das Nexus Projekt: **Umgebungsmodelle für mobile, kontextbezogene Systeme.** Projekt-Webseite, Stand 2005 `http://www.nexus.uni-stuttgart.de`

[NGS+01] D. Nicklas, M. Grossmann, T. Schwarz, S. Volz und B. Mitschang: **A model-based, open architecture for mobile, spatially aware applications.** Proceedings of the 7th International Symposium on Spatial and Temporal Databases, SSTD 2001

[NGS03] D. Nicklas, M. Grossmann und T. Schwarz: **NexusScout: An Advanced Location-Based Application On A Distributed, Open Mediation Platform.** Proceedings of the 29th VLDB Conference (Demonstration track), Berlin, Germany, 2003

[NHM+04] D. Nicklas, N. Hönle, M. Moltenbrey und B. Mitschang: **Design and Implementation Issues for Explorative Location-based Applications: the NexusRallye.** Proceedings for the VI Brazilian Symposium on GeoInformatics (GeoInfo), November 22-24, 2004

[NPM01] D. Nicklas, C. Pfisterer und B. Mitschang: **Towards Location-based Games.** A. Loo Wai Sing, Wan Hak Man, Wong Wai, C. Tse Ning (Hrsg), Proceedings of the International Conference on Applications and Development of Computer Games in the 21st Century (ADCOG 21) Hongkong Special Administrative Region, China, November 22-23 2001

[NM01] D. Nicklas und B. Mitschang: **The Nexus Augmented World Model: An Extensible Approach for Mobile, Spatially-Aware Applications.** Y. Wang, Yingxu, S. Patel, R. Johnston, Ronald (Hrsg), Proceedings of the 7th International Conference on Object-Oriented Information Systems (OOIS), Calgary, Canada, August 27-29, 2001

[NM04] D. Nicklas und B. Mitschang: **On building location aware applications using an open platform based on the NEXUS Augmented World Model.** Software and Systems Modeling, 2004

[Nic+00] D. Nicklas et. al. **Final Report of Design Workshop.** Universität Stuttgart: Center of Excellence 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Fakultätsbericht Nr. 2000/08, 2000

[Observations] S. Cox (Hrsg.): **Observations and Measurements.** Open GIS Consortium Inc., OGC 03-022r3, Version: 0.9.2, Stand 4.2.2004
`http://www.opengis.org/docs/03-022r3.pdf`

[OGC99] OpenGIS Consortium: **OGC Abstract Specification, Topic 8: Relationsships between Features, Version 4.** Project Document NmbeR: 99-108r2, 26.2.1999
`http://www.opengis.org/techno/abstract/99-108r2.pdf`

[OpenWave] Openwave: **Openwave Webseite.** Stand 2002
`http://www.openwave.com`

[OWL] S. Decker, D. Fensel, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider und L. Stein: **OWL web ontology language reference.** W3C Working Draft, 31. März 2003, online verfügbar, 2003
`http://www.w3c.org/TR/2003/WD-owl-ref-20030331`

[PV02] Y. Papakonstantinou, V. Vassalos: **Achitecture and Implementation of an XQuery-based Information Integration Platform.** Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, Vol. 25, Nr. 1, March 2002

[Pas97] J. Pascoe: **The Stick-e Note Architecture: Extending the Interface Beyond the User.** Proceedings of the International Conference on Intelligent User Interfaces. 1997

[PB99] A. G. Perez und V. R. Benjamins: **Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods.** Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5), MorganKaufmann, 1999

[PT02] W. Piekarski und B. Thomas: **ARQuake: The Outdoors Augmented Reality System.** Communications of the ACM, vol. 45, no. 1, Seiten 36-38, ACM, Januar 2002

[Pfi01] C. Pfisterer: **Konzepte für mobile, ortsbasierte Spiele.** Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1818, 2001

[RNC+02] V. Raman, I. Narang, C. Crone, L. Haas, S. Malaika, T. Mukai, D. Wolfson und C. Baru: **Data Access and Management Services on Grid,** 5th Global Grid Forum (GGF5) Workshop, 2002
http://www.gridforum.org/Meetings/GGF5/papers.htm

[RC89] D. A. Randell und A. G. Cohn: **Modelling Topological and Metrical Properties in Physical Processes.** Proceedings of the First International Conference on the Principles of Knowledge Representation and Reasoning Processes, 1989

[RDF] D. Beckett und B. McBride: **RDF/XML Syntax Specification (Revised).** W3C Proposed Recommendation 15. Dezember 2003, online verfügbar, 2003
http://www.w3.org/TR/rdf-syntax-grammar

[RSZ+04] C. Rocchi, O. Stock, M. Zancanaro, M. Kruppa, und A. Krüger: **The Museum Visit: Generating Seamless Personalized Presentations on Multiple Devices.** N. Jardim Nunes und C. Rich (Hrsg), Proceedings of the International Conference on Intelligent User Interfaces (IUI), ACM Press 2004

[RC00] M. Roman und R.H. Campbell: **GAIA: Enabling Active Spaces,** Proceedings of the 9th ACM SIGOPS European Workshop, Kolding, Denmark, 2000

[RBB03] K. Rothermel, M. Bauer und C. Becker: **Digitale Weltmodelle – Grundlage kontextbezogener Systeme.** Total Vernetzt?!, Springer-Verlag, 2003

[SensorML] M. Botts (Hrgs.): **Sensor Model Language (SensorML) for In-situ and Remote Sensors.** Open Geospatial Consortium Inc., OGC 04-019r2, Version: 1.0.0 beta, 1.11.2004
https://portal.opengeospatial.org/files/?artifact_id=7927

[Sat96] M. Satyanarayanan: **Fundamental Challenges in Mobile Computing.** Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996

[SAW94] B.N. Schilit, N. Adams und R. Want: **Context-Aware Computing Applications.** IEEE Workshop on Mobile Computing Systems and Applications, 1994

[SGML] **SGML.** ISO 8879:1986 Information processing – Text and office systems – Standard Generalized Markup Language (SGML)

[SFS] Open GIS Consortium, Inc. **OpenGIS Simple Features Specification For SQL, Revision 0.** Well Known Text, 8. September 1997

[SHG+04] T. Schwarz, N. Hönle, M. Grossmann und D. Nicklas: **A Library for Managing Spatial Context Using Arbitrary Coordinate Systems.** Workshop on Context Modeling and Reasoning (CoMoRea) Workshops-Proceedings of the 2nd IEEE Conference on Pervasice Computing and Communications (PerCom), Orlando, Florida, March 14-17, 2004

[SIG+04] T. Schwarz, M. Iofcea, M. Grossmann, N. Hönle, D. Nicklas und B. Mitschang: **On Efficiently Processing Nearest Neighbor Queries in a Loosely Coupled Set of Data Sources.** Proceedings of the 12th ACM International Symposium on Advances in Geographic Information System (ACM GIS 004), Washington D.C., November 12-13, 2004

[Sch04] M. Schumacher: **KontextWiki.** Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Diplomarbeit Nr. 2214, 2004

[Som95] I. Sommerville: **Software Engineering.** Addison-Wesley Publications, 5. Auflage 1995

[Sow00] J. F. Sowa: **Knowledge Representation.** Brooks/Cole, 2000

[SQL99] International Organization for Standardization: **Database Language SQL.** Document ISO/IEC 9075:1999, 1999.

[SGH+99] N. A. Streitz, J. Geißler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, P. Seitz und R. Steinmetz: **i-LAND: An interactive Landscape for Creativity and Innovation.** ACM Conference on Human Factors in Computing Systems (CHI'99), Pittsburgh, Pennsylvania, USA, May 15-20, 1999

[UDDI] **Universal description discovery and integration (UDDI).** Stand 2001
`http://www.uddi.org`

[UPnP] UPnP Forum: **Universal Plug and Play (UPnP) Device Architecture.** Version 1.0.1, 2. Dezember 2003

[URI] T. Berners-Lee, R. Fielding und L. Masinter: **Universal Resource Identifiers (URI): Generic Syntax.** Network Working Group, Request for Comments: 2396, August 1998.
http://www.ietf.org/rfc/rfc2396.txt

[UG96] M. Uschold und M.l Gruninger: **Ontologies: Principles, Methods and Applications.** Knowledge Engineering Review, 11(2), June 1996

[WGS84] EUROCONTROL: **World Geodetic System 1984.** Webseite, Stand 2005
http://www.wgs84.com

[VGH+02] S. Volz, M. Grossmann, N. Hönle, D. Nicklas und T. Schwarz: **Integration mehrfach repräsentierter Straßenverkehrsdaten für eine föderierte Navigation.** it+ti. Bd. 44(5), 2002

[WDC+04] X. Wang, J. Dong, C. Chin, S. Hettiarachchi und D. Zhang: **Semantic Space: An Infrastructure for Smart Spaces.** Pervasive Computing, IEEE, Juli-September 2004, Vol 3, Nr. 3

[Wikipedia] Wikipedia: **Die freie Enzyklopädie.** Stand Januar 2005
http://www.wikipedia.org

[WSDL] E. Christensen, F. Curbera, G. Meredith und S. Weerawarana: **Web Service Description Language 1.1 (WSDL).** W3C Note 15 March 2001
http://www.w3.org/TR/wsdl

[WZG+04] X. H. Wang, D. Q. Zhang, T. Gu und H. K. Pung: **Ontology Based Context Modeling and Reasoning using OWL.** Proceedings of 2nd IEEE Annual Conf. on Pervasive Computing and Communications (PerCom), Workshop Context Modeling and Reasoning (CoMoRea), IEEE Computer Society, 2004

[XLink] W3C: **XML Linking Language (XLink) Version 1.0.** W3C Recommendation 27 June 2001
http://www.w3.org/TR/xlink

[XML] T. Bray, Jean Paoli, C. M. Sperberg-McQueen, E. Maler und F. Yergeau: **Extensible Markup Language (XML) 1.0.** Third Edition, W3C Recommendation 04 February 2004
http://www.w3.org/TR/2004/REC-xml-20040204

[XML Schema] W3C: **XML Schema Part 0: Primer.** W3C Recommendation, 2 May 2001
http://www.w3.org/TR/xmlschema-0

[XQuery] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie und J. Siméon: **XQuery 1.0: An XML Query Language.** W3C Working Draft 29 October 2004
http://www.w3.org/TR/xquery

[Zha05] Y. Zhang: **Dienstmodellierung in Nexus.** Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Diplomarbeit Nr. 2238, 2005

[ZXL02] B. Zheng, J. Xu, D. L. Lee: **Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments.** IEEE Transactions on Computers, Vo. 51, 2002