Department of Computing
Imperial College of Science, Technology and Medicine
University of London

Supporting Location-Awareness in Open Distributed Systems

Ulf Leonhardt

May 1998

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering of the University of London, and for the Diploma of Imperial College of Science, Technology and Medicine.

Abstract

Mobile computers and communication devices are establishing themselves as ubiquitous features of daily life. This development is linked to tremendous growth in the number and sophistication of mobile and mobile-aware software applications. Increasingly, such applications need access to information about their own and other objects' physical locations, a requirement known as *location-awareness*.

Existing location-aware applications and systems are typically tailored to a particular type of positioning technology. This is unsatisfactory considering that no technology provides ubiquitous coverage. Equally, there are few accepted models and abstractions for building location-aware applications, making their design and implementation costly and error-prone.

Location-awareness raises legitimate concerns about personal and organisational privacy. These vary widely across administrative and application domains. Hence, there is a need to find a model allowing a balance between protection and functionality as appropriate for a particular target environment.

The functionality and structure of a location-aware system fundamentally depends on the location model employed. In this thesis, we argue that a formally specified location model facilitates design and implementation of such systems. Also, location-awareness generally cannot be achieved autonomously but requires support by a *location service*, tracking physical location of objects, and optionally providing location prediction, access control and other functions. Inevitably, location service and location model are closely intertwined.

This work presents a model of location information, the components of a location service to support it, and an architecture-independent protection model. Our approach is characterised by the following orthogonal principles:

- A global, general location service provides a ubiquitous infra-structure for location-aware applications,
- A hierarchical, semi-symbolic location model forms the basis of the service's functionality and structure,
- Location privacy is protected by policy-based access control.

The applicability of the model is supported by an implementation of a general location service prototype.

Acknowledgements

I would like to thank my supervisor, Jeff Magee, for inspiring and guiding me during the research described in this thesis. Special thanks are due to Steve Crane, who profoundly shaped my understanding of distributed systems, and who provided invaluable help during the preparation of this manuscript. Further, I would like to thank Emil Lupu for many fruitful discussions, and for teaching me snooker and squash.

I owe a great debt of thanks to Anthony Finkelstein, Erika Horn, Jeff Magee, and Morris Sloman for helping to secure the funding which enabled this research. Further thanks are due to Matthias Radestock, Sue Eisenbach, Anthony Finkelstein, and Jeff Magee for making me stay at Imperial College much longer than I had originally planned. Also, I would like to thank all members of the Distributed Software Engineering group for providing an atmosphere of intellectual stimulation, and for encouraging me in times of difficulty.

This work has been carried out initially with the support of the Human Capital and Mobility programme of the European Union (contract number ERBCHB1CT941657) and later the Multimedia and Networking Applications programme of the EPSRC (grant number GRL06010).

I would like to express my gratitude to my parents for their continuous and unconditional support. Finally, thanks are due to Mylène for her patience, love and cooking.

List of Abbreviations

AMS	Active Map Service
API	Application Programming Interface
AOA	Angle Of Arrival
BSC	Base Station Controller
BST	Base Station Transmitter
CDPD	Cellular Digital Packet Data
DAP	Directory Access Protocol
DIB	Directory Information Base
DN	Distinguished Name
DNS	Domain Name System
DGPS	Differential GPS
DSA	Directory Service Agent
DUA	Directory User Agent
EPSRC	Engineering and Physical Sciences Research Council
GLB	Greatest Lower Bound
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HLR	Home Location Register
IP	Internet Protocol
ISO	International Standards Organisation
ITU-T	International Telecommunication Union — Telecommunications
KTH	Kungl Tekniska Högskolan (Royal Institute of Technology)
LD	Location Directory
LDAP	Light-weight Directory Access Protocol
LQS	Location Query Service
LSAA	Location Service Application Agent
LSSA	Location Service Service Agent
LUB	Least Upper Bound
MSC	Mobile Services Switching Center
OD	Object Directory
OSI	Open Systems Interconnection
PARC	Paolo Alto Research Center
PCS	Personal Communication Services
PLR	Partial Location Register
POR	Partial Object Register
PPS	Precise Positioning Service
QoS	Quality of Service
RDN	Relative Distinguished Name
RPC	Remote Procedure Call
SA	Selective Availability

SPS	Standard Positioning Service
TCP	Transmission Control Protocol
TDOA	Time Difference Of Arrival
TOA	Time Of Arrival
VLR	Visitor Location Register
VLSI	Very Large-Scale Integration
WGS84	World Geodetic System 1984

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Recent technological advances have made it feasible to measure and track the location of people, computers, and practically any other object we care about. Today, there exist a number of deployed large-scale positioning systems, for example the Global Positioning System (GPS). Positioning and tracking systems are likely to become even more ubiquitous in the future. Equally, the increased mobility of people and computers has created a growing demand for location information. Location-awareness is becoming an essential feature of software applications, especially for those applications targeted at mobile end-users. Moreover, location-awareness enables new kinds of services and applications.

While demand and supply are in place, what is lacking is some kind of platform or infrastructure to build location-aware systems. Currently, such systems are mostly focussed either on a particular application or on a specific sensor technology: there is no consensus on common abstractions for designing those kinds of systems. As a consequence, there is no general service infrastructure on which to build location-aware software. Hence, it is extremely difficult to develop a location-aware application without making assumptions about the underlying sensor technology.

In this thesis, we propose a conceptual framework for dealing with location information. In particular, we propose data models and processing abstractions to be used by location-aware systems. On the basis of these abstractions, we propose a security-enabled service infrastructure as a platform for building location-aware solutions.

To prepare the reader for the coming chapters, we outline below some of the starting points of our reasoning: location-awareness, privacy, and open distributed processing.

1.1.1 Location-awareness

Interaction within a spatially distributed system is, typically, subject to distance constraints. Therefore, a mobile object's physical location fundamentally affects its ability to interact with other system components. For example, a mobile telephone works only if a base station is within range. Although a mobile object's ability to interact can be directly observed in some cases, location cannot generally be ignored in the presence of mobility. In other words, mobility naturally leads to the need for location-awareness.

Location-awareness is often directed inwards: a mobile user wants to know where he or she is. Equally, it may be directed outwards: parents would like to know where their children are, secretaries may be required to keep track of their bosses. In other applications, it may not be obvious to the users that a program is location-aware. They may simply realise that they do not have to reconfigure their environment when they have moved to a new location. Indeed, location-awareness appears to be a key to providing location transparency.

Generally, location-awareness facilitates an application's awareness of its environment or context. However, location-awareness does not necessarily imply context-awareness, and context-awareness may be achieved without location-awareness.

Location-awareness carries its cost. Outward location-awareness normally requires mobile communications at least on a local scale. Inward location awareness may be achieved through an attached location sensor. Further, there is additional design and implementation complexity, which is not yet addressed by reusable abstractions and services.

1.1.2 Privacy

A person's activities are typically correlated with the locations where those activities are performed: in the kitchen, people prepare meals, in the gymnasium, people work out, etc. In effect, knowledge of a person's location drastically reduces the uncertainty about what they are doing. This uncertainty decreases further if it is also known who a person is with. For example, somebody seeing a lawyer in his office is likely to discuss some legal matter. Location tracking can provide a (literally) far-reaching and detailed picture of a person's activities.

Hence, it is only natural for people to demand privacy, that is, control over the access to information regarding their own location. Equally, certain locations may require special privacy guarantees. For example, a company might decide that nobody should be able to see who is in the boardroom.

Current security models for location information are mostly tailored to the needs of a specific system. This often implies a limited flexibility and functionality of the security model. Also, an ad-hoc system easily introduces dependencies that limit re-use.

1.1.3 Open Distributed Systems

The defining characteristics of an open distributed system are *distribution* and *openness*. Distribution entails that the system's components reside at different physical locations. Openness means that components may enter or leave the system. Typically, openness is achieved by components having well-defined and published interfaces that support interaction through well-defined and published protocols.

A common architectural pattern of open distributed systems is the client-server paradigm. At the centre of this pattern is the concept of a service — a resource or function that is provided, not to a particular consumer, but to anybody who fits into the consumer role. A system component providing services is called a server component.

At the moment, there is no universally agreed set of services and server components to support location-aware applications and services.

1.2 Requirements

Support for location-awareness in an open distributed system must meet the requirements from the following classes:

Location model The location model should provide an expressive, flexible, and efficient representation for the locations of mobile objects. Only essential knowledge of spatial properties of locations should be represented. Further, the model must be independent of application domain and location sensor technology. Also, the model must be suitable for graphical representation.

Architecture A support platform for location-awareness must be open to heterogeneous sensor systems and to heterogeneous applications. Also, it must scale in terms of the number of applications, the geographical scope, and in the number of mobile objects. It is also desirable to compose the architecture from a number of independent building blocks that can be re-arranged to meet different sets of target requirements.

Security Location secrecy and location-awareness are conflicting requirements. The acceptable tradeoffs depend on external constraints. Hence, the platform must allow fine-grained control of this balance in order to ensure privacy of individuals and organisations. The security model must not make assumptions about sensor-technologies or the deployment environment that would limit its generality.

1.3 Contribution

The claim to *novelty* and *utility* of the work described in this thesis is based on the following considerations.

1.3.1 The location model

A location-aware system must be based on a well-defined location model. The major source of originality in this thesis is our semi-symbolic hierarchical location model. The model is predominantly symbolic, because location-awareness is most intuitive when using named locations. The model is hierarchic, because multi-resolution processing is the key to reducing complexity and ensuring scalability. This simple yet powerful model, which is free of immediate dependencies on sensor technologies or application domains, provides the key to solving issues of security, scalability, and manageability. This is the core theme of our thesis.

A second original solution is our approach to security for location information. The approach is based on a novel generalised access control matrix in the form of multi-target access control policies. Our model allows the selective restriction of the level of location

detail revealed to location-aware applications. This allows fine-grained control over the balance between privacy and location-awareness. Similarly, we can selectively protect people's identities by only revealing some of their roles. Our model uses the hierarchical location model and is thus free of architectural or scope dependencies.

1.3.2 The utility of the model

A model is useful if it reduces the complexity of the problem domain without omitting the information necessary to understand problems and to generate solutions. By imposing a canonical structure on the location space, we achieve such a reduction of complexity. We demonstrate that the resulting model is powerful enough to serve as the data model for location tracking and access control. For example, the model provides an elegant way of identifying and resolving conflicts between location sightings, leading to a novel algorithm for fusing location data from multiple heterogeneous sensors. The reduction of complexity achieved by the model is flexible. This allows multi-resolution processing which facilitates scalable algorithms and architectures. Finally, the utility of the location model is supported by a location service implementation.

Our location security model, in itself demonstration of the versatility of the location model, has formed the basis for a multi-level access control scheme for a location service. Because this scheme is consistently exploiting both the location model and the security model, it is highly configurable to allow for different security requirements. Its feasibility has been shown by an implementation of an access control mechanism for a location service.

1.3.3 Prototyping the model

A model cannot be trusted unless its validity and utility have been established by experiment. For this purpose we rely on two prototype implementations covering different aspects of this work:

- For more than two years we have been operating an "Active Office" location service for our research group. It acquires data from multiple heterogeneous sensors, and provides it to clients in a distributed environment. During design and construction of this system we have prototyped our data acquisition framework, as well as the concepts of our architectural approach. Further, the prototype has been instrumented in order to supply sample data for prediction metrics and heuristics.

- A database-centric location service provided a platform to implement and validate the semi-symbolic hierarchic location model and the protection model that are central to this thesis. This prototype supports our argument that a well-specified location model is essential for the design of a general location service.

Thus, the key concepts that are described in this thesis are supported by prototype implementations.

1.4 Structure

The next chapter is dedicated to the review of related work. We elaborate further the requirements faced by a support platform for location awareness. We explore how other approaches and systems meet those requirements, and where they fall short of them. It is of particular interest how design choices determine whether a given requirement can be met.

Chapter 3 describes the core theme of the work underlying this thesis: the design and use of a hierarchical location model. After an analysis of existing models, we propose and formally specify our own hierarchic location model. We support our model with a naming scheme for symbolic locations and with location-dependent predicates.

Chapter 4 examines the considerations that determine the architecture of a location service. We examine in detail the functionality that is required, and propose an architecture for a global and general support platform for location awareness.

Chapters 5 and 6 are concerned with the actual processing of location information. While this employs our location model, it is necessary to introduce the additional model of a *movement plane* to capture the characteristics of an object moving in location space. We also introduce a novel algorithm to integrate location sightings from multiple sensors.

Chapter 7 presents our approach to the security of location information. We examine traditional approaches to security and their shortcomings when applied to location information. We extend the traditional access matrix model and use this as a basis to propose a protection model for location information.

In conclusion, chapter 8, summarises this thesis and reflects how the original requirements where met. We suggest some avenues for further research in this area.

Chapter 2

Related Work

The high-level goals identified in the previous chapter are too abstract to form a sound basis for design decisions even on the architectural level. Hence, we need to identify a list of requirements, which can subsequently be refined. When judging existing approaches against those requirements, it is important to consider which constraints were imposed in order to meet particular requirements. This balance between constraints and requirements is the key to understanding the strengths and weaknesses of existing systems.

The objective of this chapter is firstly to define the requirements that should be addressed by a general location service. Secondly, we consider related work in order to identify the impact of particular design decisions on those requirements. Having done this, we redefine the objectives of this thesis.

2.1 Identification of requirements

A general support platform for location awareness, that is, a location service, needs to address end-to-end requirements from the following classes:

- *Defining requirements* have to be satisfied for the "black box" to qualify as a location service.
- *Scope requirements* have to be met if the location service is to be used as part of an open, global mobility support platform.
- *Operational requirements* are related to the quality of service provided to clients.
- *Business requirements* provide the foundations of the business model for the provision of a location service.

In the paragraphs below, we enumerate the major requirements for each category in order to create a "requirement space". Having done this, it is then possible to map the requirement space into the design space.

2.1.1 Defining requirements

- *Location information about real objects.* A location service must provide information about the physical locations of real-world objects.

- *Logical centralisation.* Location-aware applications require a *single point of access* to use *all* the functionality of the location service.
- *Near real-time information.* Location-aware applications mostly require knowledge of present locations, so timely information delivery is important.

These requirements basically define what we mean by a location service: a logically centralised entity that keeps track of the physical location of real things.

2.1.2 Scope requirements

- *Generality.* The service accommodates diverse types of location-aware applications and location sensors.
- *Openness.* Independently constructed location-aware applications and location sensors can use the service.
- *Global coverage.* Potentially, a location service could cover both all possible locations (i.e. the world) and the objects in it.

The satisfaction of scope requirements distinguishes special-purpose location services from a general and global location service. The intended scope of the location service has a considerable impact on its design: the narrower the scope, the simpler the design. A wider scope therefore requires the right abstractions to keep complexity in check. Our eventual aim is to meet the scope requirements in order to design a general and global location service.

2.1.3 Operational requirements

- *Adequate Spatial and Temporal Resolution.* Any location-aware application needs a minimum spatial and temporal resolution of information, or location-awareness will be severely limited.
- *Availability.* A location-aware application should always get the service it requires. This includes accessibility of the service and availability of location information.
- *Performance.* Low latency and high throughput are desirable for any service. This applies to both client-side and sensor-side interactions.
- *Trustworthiness.* The service should not deliver incorrect or incomplete information unless this possibility is indicated to the client. Further, the service should keep the secrets of its clients.

These requirements specify the quality of service experienced by clients. They are inherently contradictory. For example, the temporal sampling resolution may be limited by the maximum update throughput in a particular location service design. Hence, the design of a location service needs to identify acceptable tradeoffs given a concrete set of requirements and constraints.

2.1.4 Business requirements

The high-level architecture of a global service and the underlying business model are mutually dependent. Therefore some of the high-level architectural decisions require an awareness of those issues.

- The architecture must support multiple location service providers with overlapping coverage areas. Since a location service provider is unlikely to *own* its coverage area, the location space becomes a shared resource. Therefore, it is unlikely that the coverage areas of location service providers will be mutually exclusive.

- Users have to register with a location service provider in order to be tracked or to receive location information. Registration is necessary in order to relate real-life objects with data items in the location service. (This raises privacy issues that need to be addressed.)

- Interfaces and protocols are published and standardised. This reduces the engineering complexity of communication among location-service providers and also between location service providers and location-aware applications.

Ultimately, a location service is provided by one or more organisations, which need to commit resources to set up and maintain the service. Hence, it is conceivable that users would be charged in some way for using the service. While a comprehensive discussion of these issues is outside the scope of this thesis, we believe that the above features support the commercial viability of a location service provided by a community of cooperating yet competing location service providers.

2.1.5 Summary

A location service must provide a minimum functionality of tracking the physical locations of real objects in near real-time. Further, a general and global location service needs to have a wide scope in terms of sensors, applications, and spatial coverage. As a platform support service, a location service must also maintain a certain quality of service so location-aware applications can depend on it. Finally, the architecture of a global location service must facilitate service provision by a community of location service providers

2.2 Design consideration

From these end-user requirements, we infer the following design goals.

- *Scalability*. The need for a scalable design is derived from the global coverage requirement and the performance requirement.

- *Manageability*. Only a managed system can deliver long-term availability and trustworthiness. Further, the business requirements entail a certain administrative overhead (e.g. for the handling of subscription profiles), i.e. management.

- *Abstraction.* Openness and generality require a certain level of independence from specific application domains, location tracking systems, and computing environments.
- *Fault tolerance.* Since faults will happen in any system, the design must facilitate graceful degradation and recovery.
- *Distribution.* This consideration is primarily a consequence of the needs for global coverage and performance. It is also implied by scalability and fault tolerance. Also, the business requirement for multiple service providers indicates a need for a distributed and *decentralised* design.
- *Security.* In an open system, measures are necessary to protect secrecy and integrity of location data, as well as availability of the location service.

The design principles stated above must serve as a guide to arrive at solutions for the end-to-end requirements. If one or more of the goals can not be adhered to, also the end-to-end requirements need to be revised.

2.3 Related work overview

2.3.1 Systems containing similar functionality

Mobile communications networks have built-in special-purpose location tracking systems in order to set up communication channels between mobile parties. Examples are GSM [48] and Mobile IP [50]. These systems generally fall short of the requirements of generality and openness. However, they typically expose the desirable properties of scalability, fault-tolerance, and manageability.

Mobile tracking and positioning systems are special-purpose mobile networks that are used solely to facilitate location tracking or navigation. Active Badges [75] and GPS [78] fall into this category. Their prime deficiency is that they are normally tied to a narrow set of tracking technologies, thus making global coverage impossible. Further, location models are typically derived from the sensor technology. Thus there is a lack of abstraction.

Geographical Information Systems are database-centric systems that are used to store, retrieve, and analyse spatially-referenced data [80]. This functionality can (in principle at least) be extended to track the location of mobile objects. However, the design of GIS does not support real-time tracking of large numbers of mobile objects or real-time delivery of that information.

Therefore, GIS systems appear to be unsuitable to serve directly as location tracking systems. However, a location service may well have a GIS component to provide additional information about locations. Further, some of the techniques for location modelling and indexing developed for GIS are also applicable in the wider context of a general location service.

Distributed Directories Distributed naming and directory services, such as the Internet domain service DNS [46, 47] or the X.500 directory service [8] offer scalable and fault-tolerant designs to provide a directory service to a very large number of clients. Mapping names to addresses is not much different from mapping names to locations. However, directory services such as DNS do not cope very well with frequent updates. Thus our requirement of real-time information delivery is not met by those designs.

On the other hand, service trading [6], a special kind of directory function, is required to cope with frequently changing information. Scalability of traders is very much an area of ongoing research. No large-scale systems have been deployed yet.

Systems Management Research in systems management has spawned the idea of grouping managed objects into management domains in order to apply the same management policies to all members of a given domain [72]. When grouping managed objects into domains by location we can apply management policies to all objects at a particular location. However, the grouping of objects into management domains tends to be static and explicit, which is incompatible with the real-time information delivery requirement.

2.3.2 Related areas of computing science

Distributed Systems The location service is a distributed client-server system. Further, the location service itself may actually be distributed across many servers in order to satisfy the requirements of scalability and high availability.

Computer Networks The location service assumes that sources, servers, and clients of the service are connected to a computer network. Different parts of the systems require different protocols to operate effectively and efficiently.

Mobile computing The location service has been made possible and necessary only with the emergence of networked computing environments with a significant number of mobile computing devices. These mobile devices are important producers and consumers of location information.

Therefore, the location service will have components that can run on mobile hosts in order to gather location data. Also, clients of the location service can be mobile. Further, the mobile host may emulate part of the location service locally in order to provide gracefully-degraded functionality when disconnected from the communication network.

In the design of the mobile components and their communication protocols the resource constraints faced by mobile computers need to be taken into account.

2.4 Location tracking and navigation

In this section we discuss the basic technologies which are currently used to locate mobile objects, such as people and computers. We give an overview of the main technologies to build *absolute* sensors, that is sensors that do not rely on knowledge of previous positions.

Since integration and abstraction is our overall goal, we concentrate on technologies that we believe will form the basis of location sensors in the near future.

2.4.1 Cellular infrared networks

The main incarnation of this location-tracking technology is the Active Badge system (developed by Olivetti). An Active Badge location tracking system consists of a network of fixed infrared transmitters/receivers (badge sensor), and a number of mobile infrared transponding computers, or 'badges' [20].

Badges can be worn by people or attached to equipment (such as computers). Each badge periodically (typically every 10 seconds) emits a beacon carrying its own identity. If a sensor picks up the beacon, the location of the badge is known. The system typically allows for room-level resolution since infrared waves cannot pass through walls. When combined with low-energy radio fields the system can also provide more accurate measurements.

Infrared networks have the great advantage of not requiring a part of the precious radio spectrum. On the other hand, the short effective range of the transmitters makes comprehensive deployment outdoors impractical.

2.4.2 Cellular radio networks

Cell-based radio networks include the categories of cellular telephone networks (such as GSM [48]) and wireless local area networks (such as WaveLan [17], page 54).

In such systems, the cell base stations send out regular beacon signals. This enables the mobile station to monitor the signal quality of available cells. Based on the measurement of the strength of those beacon signals, the station decides when to switch to a new cell. The switching process is called *hand-off* or hand-over. During hand-off, location information on the fixed part of the network is updated if necessary.

The tracking mechanism employed by those systems is the converse of the strategy employed by the Active Badges. There, the mobile unit sends out beacons, which are picked up by the network. Here, the mobile unit listens for beacon signals and thus knows where it is. A significant distinction of the latter approach is that all the location measurements are performed at the mobile end. However, this may conflict with resource limitations on the mobile unit.

Radio cells are typically larger than the infrared cells described above, facilitating outdoor deployment, especially in rural areas with low subscriber density. However, this also means that typical radio cells can be too big for effective positioning (depending on the needs of location-aware applications).

2.4.3 Satellite-based radio navigation

Satellite navigation is used widely through the Global Positioning System (GPS [78]). GPS is a world-wide, satellite based radio navigation system. It provides three-dimensional position, velocity, and highly-accurate time information to users having GPS receivers anywhere on or near the surface of the earth [17].

GPS has been developed and is operated by the U.S. Department of Defence. It offers two levels of service — a Standard Positioning Service (SPS) and a Precise Positioning Service (PPS). SPS is available world-wide free of charge and provides the capability to obtain a horizontal positioning accuracy of within 100 meters and a vertical positioning accuracy within 140 meters. Most of this spread is due to noise (called Selective Availability, SA) introduced by the operator in order to safeguard its national interest. The PPS is a more accurate service but restricted to military use [17].

The GPS system consists of 24 satellites in six orbital planes with a 12-hour period. They are positioned so that between five and eight of them are visible from any point of the earth. Monitor Stations and Ground Antennae control them.

Principle of function Each of the satellites continually broadcasts time information derived from their internal atomic clocks along with data of their present position. The GPS receiver, which typically contains a multi-channel radio receiver, can measure the time it takes for the signal to traverse the distance from the satellite to the receiver. This method is known as *time of arrival* (TOA). If the distance to three satellites is known, it is possible to calculate the three dimensions of the receiver's position. Hence, the signal from a fourth satellite is needed to provide the exact time. If only a two-dimensional position is required, signals from three satellites are sufficient.

Coverage and Accuracy In order to get any kind of position, at least three visible satellites are required. While satellite radio signals are very resistant to interference, they are reflected from buildings and windows. Therefore, GPS cannot normally be used indoors. Further, GPS sightings tend to be more inaccurate in built-up areas since reflections may lengthen the path from satellite to receiver. In certain areas the amount of visible sky may also be restricted, thus reducing the probability of finding the required number of satellites.

Accuracy enhancements If additional information about the movement of the receiver is available (such as knowledge of the fact that the receiver is stationary), SPS sightings can be made more accurate by statistically eliminating some of the error.

Radically improved accuracy can be obtained by employing *Differential GPS* (DGPS). DGPS uses a fixed GPS receiver with known position. Therefore, the SPS noise can be identified. By broadcasting this knowledge other receivers can eliminate the SA noise from their measurements and get improved sightings. DGPS is typically offered as a value-added commercial service and employs radio broadcasts to transmit the DGPS information. DGPS cannot remedy distortions due to signal reflections or obstructed signal paths as experienced in built-up areas.

2.4.4 Terrestrial radio navigation

The positioning principle used by GPS can also be used with a set of terrestrial rather than satellite-based radio transmitters. The common techniques are time of arrival (TOA, used by GPS), angle of arrival (AOA) and time difference of arrival (TDOA) (see [85] for

a more detailed discussion). The necessary calculations can be performed either by the mobile unit or by a server on the fixed network.

This technology has recently gained momentum with the requirement of the U.S. government that mobile phones must provide a "push-button" emergency service with caller location (wireless E911). The caller must be located within a radius of 125m in 67% of all cases [85]. Since the radio cells of current mobile telephone systems are mostly bigger (300m...35km), an additional location technology is required. GPS may not be suitable since it does not work very well in built-up areas. On the other hand, the existing base stations of mobile telephone networks can be used for terrestrial radio navigation.

In the wireless E911 application, the mobile terminal would try to measure its current position with respect to three or more base stations when demanded. Then, the terminal can send the mayday message along with its coordinates to a public-safety answering point. As a result of this, mobile telephones may become aware of their own physical location, thus enabling other location-aware applications and services.

2.4.5 Monitoring of fixed terminals

The technologies described above are overkill when it comes to locating people who spend most of their time working on fixed networked computer terminals (or appliances containing computers, such as copy-machines). Some researchers predict an even greater computer penetration of our everyday environment in the future [76]. By monitoring access to those terminals, location data can be gathered cheaply, non-intrusively, and reliably. An example is the `ruser` service offered by many UNIX systems (e.g. SunOS [70]).

The UNIX operating system keeps track of logon and idle times for all terminal sessions. This information is made available via the `ruser` service. It can be queried via remote procedure calls (RPC) to provide a list of currently logged-in users along with their idle time and the name of the terminal. Since most terminals have a fixed location, a network of UNIX workstation with enabled `ruser` service can be used to build an effective location tracking system.

In order to track people's location with such a system, no additional hardware is required. This approach also offers authenticated data since the user's login session must normally be authenticated with user-name and password. On the other hand, this approach is only suitable for environments where a significant part of the population regularly accesses fixed and networked computers.

The same approach could be used to piggy-back location tracking onto other networked infrastructure with elements of authentication, such as telephone handsets or automatic teller machines. In many of these cases, concern for privacy outweighs the potential benefits.

2.4.6 Relative sensing technologies

The above technologies are sometimes referred to as *absolute* sensors since location measurements do not *require* knowledge of previous positions (although they may be facilitated by such information). *Relative* sensors measure properties of a located-object's

movements, such as speed, direction, or distance travelled. The respective devices are tachograph, gyroscope, and odometer.

The typical use of those devices is to compensate for transient unavailability of direct position measurements from absolute sensors (e.g. dead-reckoning [85], page 45). Typically, such methods have cumulative errors, that is, with time the position measurements become increasingly inaccurate. Therefore, relative sensors can only be used for positioning in permanent combination with an absolute sensor (as opposed to a one-off calibration). The resulting two-sensor systems appear like improved absolute sensors. For this reason, we largely ignore relative sensors on the higher levels of the location service while acknowledging their importance for constructing integrated multi-sensor systems. Location prediction (see chapter 6) is perhaps the main application of relative location measurements in a location service.

2.5 General location services

In this section, we describe general location *services* that have been built on top of some of the basic technologies described above.

2.5.1 Active Badge System (Olivetti)

Building on the Active Badge technology, researchers at Olivetti have proposed and implemented an architecture for a distributed location service [20]. Their design follows a client-server approach. The functionality is partitioned into services provided by the following servers:

- The *Location server* collects the badge sightings from the various sensor network controllers. It maintains a cache of the last sightings. A sighting consists of the badge address, the sensor address, and a time-stamp.
- The *Name server* offers a *White Pages* directory service that maps badge addresses and location addresses into more detailed information, such as the name of the bearer.
- The *Message server* co-ordinates message delivery to the Badges.
- The *Exchange server* controls the federation of location service between organisations. It encapsulates the issues of security, access control, and information exchange between administrative domains.

As far as the data model is concerned, the unit of location is a tuple consisting of badge address, location address, and timestamp. The location of an object is modelled as a dynamic attribute of the object, which is implemented by a pointer to a service interface for that object.

Location-aware applications built for this system include GUI teleporting [54, 79], location-sensitive communications, and location-oriented paging.

Discussion As far as our requirements are concerned, the Olivetti system is tied to the Active Badge location system. Thus, abstraction is lacking. Further, the system is targeted towards a federation of small-to-medium sized organisations. Hence, the issue of scalability has not been addressed sufficiently.

2.5.2 Ubiquitous Computing (Xerox)

Researchers at the Xerox Paolo Alto Research Center (PARC) have explored the vision of ubiquitous computing, constructing environments with ubiquitous networked computing devices [76]. Both mobile and fixed devices have been used, including Active Badges, Tabs, Pads, and Whiteboards [77]. Context-awareness supported by a location service is a central theme of this research.

The location service employs a number of tracking subsystems, including an Active Badge server and a UNIX location server [66]. *User agents* gather the information provided by those subsystems to allow for user-centric operations. They support the more abstract location services that have been constructed: Location Query Service (LQS) and Active Map Service (AMS, [59]). The location services support the new category of *context-aware applications* [58].

User Agents To allow for user control over location information, the location service is built upon a decentralised community of agents [66]. All location information concerning a particular user is gathered and stored by an agent, responsible for integrating and fusing location data. It is also responsible for registering with the appropriate regional Location Broker. Further, the user agent maintains access control over location information and can thus protect the user's privacy. The user is allowed to specify access control policies to specify the amount of trust the user agent places in centralised servers. The user agent provides two services: location of the user and ubiquitous message delivery to the user.

Location Query Service While the community of user agents is responsible for user-centric operations, the Location Query Service (LQS) provides location-centric services. The LQS is organised by region, with a Location Broker assigned to each region. The Location Broker maintains a set of references to the located-objects in its area. These references can be anonymous if the user-agent wishes to retain access control. Agents can also choose to delegate their access control to the LQS to improve efficiency.

Active Map Service The Active Map Service (AMS) provides location information in an abstract hierarchical location model. The service relies on user agents and LQS as described above.

Each server contains an active map consisting of a hierarchy of locations with a containment relation (e.g. Room-Floor-Building-Region). The area covered by a single server is a *region*. There is no location service covering more than a region, that is, clients must directly access all regions they are interested in [59].

Each server maintains a set **P** of publications (i.e., *located-objects*, their locations, and further useful information), and a set **S** of bandwidth-limited subscriptions. Both user-

centric queries and location-centric queries are supported. The dissemination of subscription updates to multiple clients is performed efficiently using multicast channels.

Context-aware applications Context-awareness enables an application to adapt to changes in its environment and location. Thus, context-awareness supports personal mobility, application mobility, and host mobility.

Important aspects of an application's context include location, nearby users, nearby resources (such as terminals, the coffee trolley, etc.), and characteristics of environment variables (light, noise, bandwidth of network connection) [58]. Context-aware applications can exploit this information in a number of ways:

- *Proximate Selection* is a user interface technique to support the choice of easily accessible objects by the user. Possibilities include annotated menus such that the user can make an informed choice, and virtual objects that are mapped automatically depending on the user's location.
- *Automatic Reconfiguration* of the application is caused by changes in the context of use. When applied too frequently, this may lead to performance problems and confusion of the user.
- *Contextual Queries and Commands* have a location-dependent semantic (e.g. print on the nearest printer, list all the people in this room).
- *Context-triggered Actions* are policies that declare what should happen upon a change in the context of use. They are adaptable to the requirements of individual users.

Context-dependent information is provided by regional *environment servers*. User-dependent information (including the location) is queried from the user agent. The application is responsible for subscribing to the appropriate environment servers. Typically, it will always subscribe to the user agent and then select the environment server according to the user's location. The abstraction of environment variables is used to provide application access to contextual information, along with a call-back facility. The asynchronous notifications implemented by those call-backs efficiently support the applications' ability to react quickly to changes in their environment [60].

Discussion This system meets our requirements for generality, openness, and real-time information. The issues of performance and trustworthiness have been addressed. Users may trade privacy for functionality and efficiency. Further, the decentralised approach offers a reasonable basis to meet our business requirements. An important feature is that location information is treated as fundamentally uncertain (with important implications for location-aware applications).

However, the system has been aimed at a campus-type environment with ubiquitous personal computing resources. Thus, the design needs further work to be suitable for global coverage of users and locations. We believe, that optional pooling of user agents into servers may be necessary to achieve this (a view supported by [29], page 4). Further,

logical centralisation currently stops at the regional level. The location model does not allow for overlapping locations, which is convenient for the service but may not be flexible enough for applications.

The proposed class of context-aware applications shows that location-awareness is an essential pre-requisite for context-awareness. The actual location-awareness may be hidden from the context-aware application if a location-aware service (such as the environment service) is used.

2.5.3 Active Office Project (Canterbury)

Researchers at the University of Kent in Canterbury have proposed the use of a *master location system* to integrate several types of local location information sources behind a uniform interface [55].

For our discussion, the following aspects are important:

- *Location representation.* A location is a list of attributes, where each attribute consists of a key (i.e. name) and a set of values. Two locations are taken to be *equal* if they have at least one key in common and the intersection of both value sets for each common key is non-empty ([55], page 5). Hence, we are faced with a flat (i.e. non-hierarchic) location model, although the basic representation would support an inclusion ordering.

- *Architecture.* The system architecture is recursive, i.e. a Master Locator can serve as a Slave Locator to another Master Locator. Thus the system can be scaled to a larger coverage area and user population. There is a location directory that stores a complete description for all locations in the system. This directory is used to expand those sightings which do not have all the attributes that belong to the corresponding location.

- *Uncertainty.* Location information is given with a confidence level (0..100). Locators can return multiple locations for the same objects, typically with decreasing confidence.

- *Location algorithm.* The location algorithm is executed by all Master Locators to retrieve and unify location information from the Slave Locators. The algorithm uses a confidence-ordered priority queue of locations. Different sightings reporting the same location are corroborated into a single queue entry. The elements of the queue are made available to the client starting at the entry with the highest confidence value.

- *Access control.* Access control is implemented using capabilities that allow access to certain Locators (e.g. Active Badges, Diary) for certain individuals. In principle, those capabilities are flexible and powerful enough to support higher-level access control mechanisms, such as policy-based access control.

- *Location strategies.* User-specific location strategies can be defined that determine which Slave locators should be queried first and what level of confidence is sufficient to report a location.

Discussion The architecture of the Canterbury location service meets our requirements for openness, abstraction, logical centralisation, and potential global coverage. Real-time information delivery is compromised by the lack of asynchronous information delivery. The system addresses trustworthiness by using access control and confidence values. Performance and availability have not been discussed explicitly. From a business perspective, the design is perhaps too centralised to permit the federated business structure we have in mind. The location model is not hierarchical and thus lacks true scalability. This is a fundamental weakness of the system.

2.5.4 The Walkstation II project (KTH Stockholm)

The Walkstation II research project [43] addresses various issues of VLSI-design, radio communication, network protocols and mobile application services. Here, we shall focus on the latter.

In contrast to the ubiquitous computing approach, the Walkstation project focuses (as the name suggests) on host mobility [42], which is supported by *floating agents* on the backbone network. Those agents try to minimise the performance impact of migration by pre-allocation of resources and pre-fetching of cache contents. This approach relies on accurate mobility models to predict future location of located-objects with a high degree of confidence.

Liu [37] proposes two kinds of mobility model: deterministic state machines (sequences and loops) to predict regular movement, and Markov chains to model constitutional constraints (highways, bridges, etc.) Both employ state machines to model movements of the mobile.

The mobility prediction algorithm uses a dynamic movement-pattern database which is matched against the sequence of **N** states in a state queue. The algorithm consists of two parts:

1. Updating the pattern database by either replacing the least recently used pattern or by increasing the weighting of an existing pattern.

2. Matching the pattern database and the Markov Model with the state queue. State sequence matching, time matching, and frequency matching are employed to select a best-match continuation trace. The continuation trace is used to predict the next state.

Simulation results show a correlation of 95% between prediction ratio and movement regularity [39].

Discussion While the Location-Sensitive Information Management component of the framework ([37], page 61) has not been elaborated sufficiently to be considered as a location

service in its own right, the proposed mobility prediction function offers a set of useful tools to improve the availability of location information. However, the solution described by Liu is only viable with near-perfect historical location data. Further, effective predictions can only be made for the immediate future.

2.6 Special-purpose location services

Today, there are already a number of location-aware systems in operation. Some of them cover large areas and large numbers of located-objects. Those systems typically contain logical or physical modules that can be classified as a special-purpose location service. In this section, we examine the salient features of those location service modules.

2.6.1 GSM

The GSM mobile telephone system contains a location management component that keeps track of subscribers' locations. The primary purpose of the location management function is to help with the setup of mobile-terminating calls ([48], page 44). The wider category of mobility management also includes authentication and security functions [30].

The cellular network is subdivided into location areas consisting of several cells each. Each cell can have a radius between 300 meters and 35 kilometres. On the network level, location information is the address of a particular location area.[1] To find out the exact location of a mobile unit the entire location area must be paged.

Each location area has a Visitor Location Register (VLR) that contains copies of the profiles of all mobile subscribers currently registered in the location area. Typically, the VLR is co-located with a Mobile services Switching Center (MSC), and both MSC and VLR cover the same location area. Additionally, each GSM network has a logically centralised subscription database, the Home Location Register (HLR). The HLR stores the network address of the most recently used VLR for each subscriber.

Location updates are necessary whenever the mobile unit enters a new location area (e.g. moves between location areas or powers up). The mobile unit listens for beacon signals of nearby cells. The *best* cell is chosen according to a well-defined metric (see [48], page 453). The mobile then tries to send an update message to indicate its new location. This update message is sent from the mobile unit to the MSC, which updates the VLR and HLR. The HLR subsequently informs the previous VLR to cancel the registration there. For reliability reasons, there is also a periodic update procedure, which is controlled by the operator. The location update procedure also includes authentication of the mobile unit and transfer of subscription data from the HLR to the new VLR.

Location information is only accessible to the network operator via the signalling network (typically Signalling System No. 7). Subscribers must trust the operator to guard their personal data.

[1] The VLR may have additional location information.

Discussion Evidently, GSM is a viable system for mobile telecommunications. However, the immediate applicability for a more general location service is limited by a number of factors:

- GSM offers an architecture built on very narrow assumptions about required functionality (find a subscriber).
- Scalability is limited because the location server hierarchy has only two layers.
- The location information is too coarse-grained for many location-aware applications.
- The network operator may not allow access to internal location data.

2.6.2 PCS location directories

It is anticipated that future personal communication services (PCS) will have much smaller cell sizes and much increased number of customers than the current GSM implementations. It is argued that this will lead to query and update volumes several magnitudes higher than in GSM systems [73]. Hence, there has been a lot of research into the efficient and scalable tracking of mobile users in future PCS networks.

As described in [3], such systems contain two basic operations: "move" (location update), and "find" (location query). Typically, proposed solutions use a multi-level hierarchy of location servers, e.g. [3, 29, 4, 26, 73]. Each location server node has a well-defined network coverage area. Location updates are triggered by the leaf nodes, and are propagated through the directory following a well-defined algorithm. In contrast, queries can generally be directed to nodes at any level in the hierarchy. This often leads to recursive query patterns.

The basic mechanisms employed to improve response time and reduce network traffic are data replication and forwarding pointers. Data replication reduces query latency and query traffic but increases update traffic. Further, the consistency of the replicas needs to be controlled. Forwarding pointers are hints about a likely location of the mobile unit. They can be traversed if the desired information cannot be found locally. Typically, forwarding pointer are left behind at a mobile user's old location if the new location is known. Consistency control is normally achieved by simple timeouts. Since forwarding pointers offers fewer guarantees than replicas, they are also more light-weight in their implementation. However, forwarding pointers do add an additional layer of complexity. Problems include pointer loops and infinite pointer chases (if the mobile user moves too quickly). Also, long pointer chains are susceptible to failures.

Since efficient queries and updates need to make assumptions about parameters of user mobility (such as the *call-to-mobility ratio* [4]), sometimes systems can adapt dynamically to changing user characteristics. The subscriber profile would be used to store such mobility parameters.

Unlike GSM, many approaches do not rely on an HLR for location queries. The location server hierarchy subsumes the HLR's location management function.

Discussion The research on future PCS systems focuses on higher scalability and better efficiency than the GSM. As a side effect, PCS locations will be smaller and thus more useful for location-aware applications. On the other hand, PCS will still be a special-purpose, closed system.

2.6.3 Vehicle-tracking systems

Many companies are faced with the task of tracking the movements of their own vehicle fleet (e.g. truck operators), or the movements of a set of other vehicles (roadside assistance services).

Vehicle-tracking systems are often employed in computer aided dispatch scenarios. These are normally found in organisations with a large number of vehicles (trucking companies, emergency and taxi services, the military) which follow non-regular routes. Real-time vehicle-tracking enables these organisations to re-route their vehicles efficiently when faced with new or changing tasks. Since this leads to substantial productivity gains and reduces response times, vehicle-tracking is already widely used. For example, a general commercial fleet tracking system based on GPS and GSM that is currently in use in Germany and South Africa is described in [69]. Rockwell's Fleetmaster system offers similar functionality with a combination of GPS and cellular digital packet data (CDPD) [17].

Vehicle-tracking systems typically have a star-shaped architecture with a dispatch centre in the middle. The vertices are the autonomous on-board positioning systems of the vehicles. Communication between the centre and the vertices is performed using a wireless wide-area network (e.g. GSM or CDPD).

For the on-board positioning system, GPS normally is common because it is cheap and available in virtually all outdoor locations, often combined with relative sensors (e.g. gyroscope, odometer) to achieve continuous measurements [85].

On the application side, vehicle-tracking systems employ few purpose-built applications on top of the location tracking service. Often this includes a map display, and a computer-aided dispatch system.

Discussion Vehicle-tracking is of interest because it shows that autonomous positioning systems can be networked to provide a location service. However, being a niche application, vehicle-tracking falls short of our requirements of generality and openness. On the other hand, global coverage (in terms of coverage area), logical centralisation and real-time information have been addressed.

2.7 Related concepts and standards

In this section, we devote some attention to existing classes of software systems that have functional or architectural aspects in common with a distributed location service. They include distributed directories, geographical information systems, and domain-based management systems.

2.7.1 Distributed directory services

A directory maps a name into a set of attributes (or properties) [84]. In a distributed directory the data repository is partitioned and held by multiple servers. Hence, *navigation* among these servers is also necessary.

Directory services include name-based lookup (white pages service), and property-based lookup (yellow pages service) [81].

The main issues in constructing a directory service are the potential size of the directory, the frequency and nature of client requests, and dependability. Typically, a scalable and fault-tolerant directory service is achieved by employing the following techniques [63]:

- Partition the service over multiple directory servers.
- Replicate directory servers for improved availability and performance.
- Cache results of queries to improve performance and reduce server load.

Also, a directory service often needs to span multiple administrative domains. This is facilitated by partitioning the service along the lines of those domains.

The spectrum of directory designs is fairly large. Today, the Internet Domain Name System (DNS) is the most widely used distributed directory service. It is discussed in more detail below. We also outline the X.500 directory standard, which has not yet achieved ubiquity. Further, we discuss *service traders*, a specialised directory service.

DNS

The Domain Name System DNS [46, 47] is a distributed naming service. It offers a white-pages service that maps names (domain names) from a hierarchical naming scheme into a list of attributes.

The name space forms a tree, with each node being uniquely identified by its absolute path. Typically, a name's attribute list represents a list of addresses for different network protocols (typically IP address, name of the email exchange). It is also referred to as a name's DNS record.

DNS is provided through a set of name servers. Each server is assigned to a subtree of the name space. The part of the name space covered by a name server is referred to as its zone. A name server is primarily an information repository storing the zone's records and links to other name servers. If a query requires that another name server be consulted, this recursion can be handled by the client, or, optionally, by the server.

DNS uses both caching and replication to reduce query response times and to increase availability. A name server can maintain secondary copies of other zones. The maintainer of a secondary copy must query the maintainer of the primary copy (Primary Server) periodically in order to find out whether the local copy is still valid. This replica control mechanism guarantees weak (i.e. eventual) consistency. Additionally, both name servers and clients can cache DNS records. Cache consistency is controlled by a simple timeout. Again, this guarantees eventual consistency.

Discussion DNS' design goal was to provide a highly available service with distributed data maintenance and storage. Data consistency is traded in for data availability. The system cannot deal well with frequent updates because all the replica control mechanisms are based on timeouts, i.e. the updates are synchronous.

The X.500 directory service

The X.500 directory standard has been developed since 1984 by CCITT (now ITU-T) and ISO. Official standards were adopted in 1988 and 1993 (for a more detailed account, see [8]). Historically, X.500 was developed as a directory service for OSI systems. This is still evident from the use of OSI protocol stacks in the various directory protocols. However, recent developments, such as LDAP [82, 21], have made X.500 somewhat more popular in the wider Internet community by employing the ubiquitous TCP/IP underneath the directory access protocol.

Functionally, X.500 offers both name-based and property-based lookup services. Users access the directory service through a directory user agent (DUA), which in turn interrogates or modifies the local directory service agent (DSA) (The DUA plays a resolver role, i.e. it provides access to the resolver service). The DSA provides access to the directory information base [19]. Queries range from straightforward DNS-style lookups to recursive attribute-based fuzzy matching.

X.500 prescribes a tree-shaped name space formed by distinguished names (DN), where each DN is an ordered sequence of relative distinguished names (RDN). Each RDN is a set of key-value assertions [84]. That is, name space and data space are the same. The DN are used to uniquely identify the objects in the directory information tree. Each object has a set of typed attributes. All the objects together form the directory information base (DIB).

The name space can be partitioned into domains (coherent subsets), with a domain maintained by a DSA. Each DSA also stores the links to all DSA serving neighbouring domains. Thus, the DSA are organised in a knowledge information tree.

DIB data can be replicated using a master-copy scheme providing weak consistency [8]. The granularity of replication can range from a single attribute of a single entry to a complete naming context. Protocols are specified for creating and managing a shadowing relationship, and for transferring updates from supplier to consumer. In order to avoid the master-copy becoming a bottleneck, shadow copies may propagate updates to secondary shadow copies. Updates can be initiated by either the supplying or the consuming DSA, with no guarantees that such a request will be satisfied. Updates can be initiated immediately, periodically, or randomly.

In addition to "static" replication, caching of query results can take place in both DUA and DSA. Since full access control information is not available in the DUA and the caching DSA, the caching of query results must take place on a per-user basis. This, of course, severely limits the effectiveness of caching if there is a high user turnover.

There have been proposals to use the X.500 infrastructure as the basis for a location information service [40]. X.500 was designed for slowly changing directory information, hence update frequency and the lack of update notifications are the obvious problem

areas. In [40], the authors describe a modified DSA which has dynamic external attributes and spatial matching rules. The dynamic attributes are queried from an external service whenever they are needed. This removes the need for frequent updates. On the other hand, replication of the dynamic attributes to unmodified DSAs seems to become impossible.

Discussion X.500 is more powerful and complex than simple name services, such as DNS. Although widely deployed, the stated goal of global coverage has not (yet) been achieved. Therefore, it is hard to evaluate whether the design is adequate. For our discussion, the meta-architecture (i.e. DSA, DUA, etc.) and the query processing facilities are of interest. Unfortunately, X.500 does not allow for real-time information delivery.

The X.500 directory access protocols (DAP, LDAP) could be used to provide query-based access to location information, although hierarchical data types would have to be emulated by multi-valued attributes. As argued in [40], such a solution would be beneficial because it uses a widely accepted protocol standard. However, there is no support for push-style event dissemination in those protocols. Hence, a location service would appear to require additional access protocols complementing the functions offered by DAP or LDAP.

Service traders

A trader is a directory service that enables clients to find suitable servers in a distributed system [6]. Both white-page and yellow-page services may be supported.

A trader accepts service *offers* from exporters of services, and service *requests* from importers of service request. To satisfy a request the trader typically has to perform attribute-based matching over the available service offers. Traders can share information about service offers amongst each other, a concept known as *federation*. That is, traders can act recursively as clients to other traders [14].

In the federation scenario, the traders form a graph, which is traversed when the federation is queried. Replication techniques and traversal algorithms are implementation-specific. The management of the federation graph is also an issue. Additionally, a hierarchically-structured *context space* may be used to facilitate trader federation. Then, imports and exports can occur relative to a *context* rather than to a specific trader [63].

There has been research on building a distributed trading function on top of the X.500 directory service [2, 51]. The idea is to use the directory information base to store the trader data (service offers, trader graph etc). The trader itself uses a DUA to access and to search this information. Further, the directory can be used to find other traders (ideally other service offers, but this seems to be difficult). The trader link graph is completely independent of the directory information tree. Thus, link traversal is implemented outside the directory (but the directory's name resolution service can be used).

Discussion Service information can be as dynamic as location information. Hierarchical context-spaces and federation graphs are valuable concepts to achieve scalability in the presence of decentralisation *and* dynamic information. However, most of this research appears to be still work-in-progress.

Directory service summary

A location service can be considered as a specialised directory service. Existing designs of directory services, such as DNS, expose the desirable properties of global coverage, openness, and logical centralisation. Some systems, such as X.500, are very general and have powerful query facilities. However, existing directory service designs cannot cope with real-time information because of the replica control mechanisms. This indicates the need for a replica control scheme tailored to the specific nature of location data.

2.7.2 Management domains

As described in [65, 72], management domains offer a framework to group objects explicitly in order to apply a common management policy. Since domains can contain other domains, it is possible to construct domain hierarchies. Domains are a generalisation of tree-structured directories. The domain space is maintained by a, possibly distributed, domain service.

Domains are used to specify subject scopes and target scopes for management policies. Thus, objects can be managed independently of their identity according to their domain membership. Typically, a policy would link a source domain to a target domain while specifying a set of permissible or mandatory actions [44]. Management policies can be specified in terms of rules. A domain can inherit policies from its parent domains. Thus, the domain structure implicitly supports policy decomposition.

Management typically occurs *within* an organisation, and co-operation between organisations requires special arrangements. In [72] a federation scheme is suggested, where organisations can choose to link foreign domain spaces into their local domain space. These links are called context references. They do not carry policy implications in order to guarantee the independence of both organisations.

Discussion The management domain approach offers a framework for presenting collocated objects. This may be used both for the visualisation of location information and for the management of located-objects. Further, a domain framework may be suitable for specifying security constraints involving location data. However, existing implementations, such as DOME [72] are geared towards static and explicit domain membership.

2.7.3 Geographical Information Systems

GIS systems store and process spatially referenced data. Here, we outline some of the modelling concepts and abstractions applied in GIS [71]. These are mostly related to modelling in the *cartographic space*. The real world is referred to as *geographic space*.

- A **cartographic model** is a set of map layers that are all registered with a common cartographic frame of reference.
- A **map layer** is a set of data describing the spatial variation in one characteristic of a geographical study area. The essential components of a map layer are *title, resolution, orientation*, and *zones*.

- A **zone** is a category into which geographic locations are classified on a map layer. The zones of a given map layer must be both all-inclusive and mutually exclusive in their spatial coverage. The components of a zone are: label, value, and locations. A zone can be fragmented.

- A **location** on a map layer is associated with a unique square unit of cartographic area referred to as grid square.

- A **neighbourhood** is a set of locations, each of which bears a specific distance and/or directional relationship to a particular location called the neighbourhood focus. This relationship may be specified in terms of maximum distance (circle), a range of distances (ring), ranges of direction (sectors), or combinations of distance and directions. Neighbourhoods may overlap and do not need to be all-inclusive.

Spatial GIS data can be classified into two groups: geometric data and topological data. Geometric data is represented in raster or vector format, and describes individual objects. Topological data is concerned with relations between objects, such as adjacency, connectivity, or inclusion.

Additionally, thematic GIS data consists of application-specific attributes related to geographic entities.

Discussion GIS are basically fancy static map databases. Real-time tracking of large numbers of users is outside their scope. However, GIS can serve to define and maintain the location spaces used by a tracking system.

2.8 Chapter Summary

This chapter has identified the requirements for a general distributed location service. From those requirements, high-level design goals were derived. Our examination of related work indicates that no existing design meets all our requirements.

2.8.1 Positioning technology

There exist different tracking and positioning technologies, each with their own advantages and drawbacks. No single approach can deliver the ubiquitous coverage required. Therefore a general location service must work with multiple tracking technologies simultaneously.

2.8.2 Location model

Flat location models inhibit scalability and abstraction. Hence, it appears that only hierarchical approaches, such as those used by the Active Map Service (Xerox PARC) or by GSM, can deliver scalability and flexibility. It also facilitates presentation and management, as shown by the management domain approach. A hierarchical data model may form the foundation of a specialised replication protocol as used by GSM.

2.8.3 Distributed architecture

This chapter has shown that analogous systems achieve scalability through distribution. While many of the larger systems (GSM, PCS, DNS) use hierarchical distribution approaches, others, such as Xerox PARC, have opted for a decentralised approach. This issue needs further consideration.

2.8.4 Thesis objectives

From the above, we derive the primary objective of this work. This is to develop a model for providing a general global location service based on multiple tracking technologies, along with a scalable architectural framework.

Chapter 3

Location Models

Location-awareness requires a data model that can adequately represent the locations of mobile and fixed objects. We refer to such a model as *location space*. There are two principal ways to design a location space: as an n-dimensional coordinate system, or as a set of symbols (i.e. names) with relationships between them. The former we call the *geometric* approach, the latter we refer to as the *symbolic* approach. Both groups of models have strengths and weaknesses; indeed the following section will discuss these.

To support location-awareness, a location model must address the requirements of location sensors and location-aware applications. Hence, it is instructive to review the models used there.

Below is a list of location sensing technologies with the location model employed:

Technology	Location space	Location identifier	Type of location model
Active Badges	Set of Sensors	Sensor identifier	Symbolic
GPS	Coordinate system	Coordinate tuple	Geometric
ruserd (UNIX)	Set of terminals	IP address	Symbolic
GSM	Set of location areas	MSC/VLR address	Symbolic
Radar	Coordinate system	Coordinate tuple	Geometric

Both symbolic and geometric location models are used by location sensing systems. Moreover, the coverage of the different technologies is complementary. GPS will not work indoors, while Active Badge sensors are not normally deployed in the countryside.

But what are the requirements of the application domain? In order to find out, consider the following list of domains of location-aware applications. In each domain, there are already applications that deal with static or dynamic location information.

Application domain	Location representation	Type of location model
System Management	Hierarchical domains	Symbolic
Naval navigation	Coordinate system	Geometric
Military command	Numbered grid squares	Symbolic
Intelligent Highway	Road name, heading, speed	Geom./Symbolic
Active Office	Building, floor, room	Symbolic
Postal system	Postcode, street, house, flat	Symbolic
GIS	Grid square, zone membership	Geom./Symbolic

It shows that existing applications use both classes of location models, some even at the same time. Hence, we expect the same to be the case for location-aware applications in the respective domains. Also, it appears that in many cases applications do not require geometric location data. They rather work with the more abstract notions provided by a symbolic location model.

Therefore, in order to satisfy the generality and coverage requirements identified in the previous chapter, a location service needs to accommodate both symbolic and geometric location information. In particular, it must be able to process sensor input in either representation, and perform client interactions in either representation.

The geometric and symbolic models are orthogonal representations for location information. Either can be used independently from the other, but they cannot replace each other. Hence, the challenge is to find a way for the two models to coexist.

In the remainder of this chapter, we shall identify the different types of symbolic and geometric location models. Further, we propose our own location model, which is a fusion of the two approaches. We will formally specify our location model and outline its application.

3.1 A brief taxonomy of location models

3.1.1 Symbolic models

Symbolic location models refer to locations by abstract symbols, i.e. names, such as "Room 429", "Stanford Campus", "Huxley Building", or "Canterbury cathedral". Since both locations and located-objects are represented as symbols, locations are modelled quite naturally as *sets*, and located-objects as *members* of sets. Following this approach, a located-object would be a member of a location whenever it is physically within the associated area or volume. This is the essence of a symbolic model.

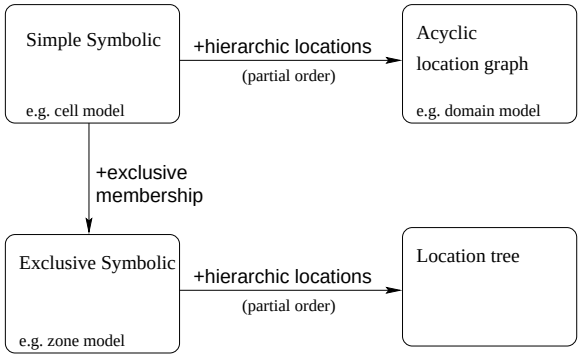


Figure 3.1: Classification of symbolic models

The set of location symbols within a given model may be constrained. For example, it is sometimes the case that locations are not allowed to overlap. Equally, the location model may include a partial ordering of location symbols based on the spatial inclusion of the underlying locations. Depending on whether locations overlap, this inclusion ordering

results in a location tree, or in a location lattice (acyclic graph). Figure 3.1 shows a classification of location models along those lines. The example models (cells, zones, and domains) will be described later in this chapter.

Advantages of symbolic models:

- Location-awareness is easier to achieve if the relevant locations can be referred to simply by name.
- Access control to location information is facilitated if the relevant location can be referred to by name.
- The hierarchical data model facilitates multi-resolution processing. This in turn can support scalability, manageability, and adaptation.
- Secondary models of location information (e.g. mobility patterns) are facilitated by symbolic location models because symbolic models are discrete and well-structured.

Disadvantages of symbolic models:

- Symbolic models require an additional layer of indirection.
- The set of useful named locations depends on the application domain. This can potentially lead to a very large number of location symbols that need to be managed.
- Symbols for locations often need to be constructed and managed manually.
- A symbolic location model restricts the spatial resolution of the location data represented in the model.

3.1.2 Geometric models

Geometric models are based on one or more reference coordinate systems. Locations are represented as points, areas, or volumes within the coordinate system. Such locations are typically described by sets of coordinate tuples. There is no inherent distinction between locations and located-objects, everything is expressed by coordinates. Therefore, geometric models are harder to apply but potentially more powerful.

Figure 3.2 shows a simple classification for geometric models. The unified geometric model contains multiple coordinate systems with (possibly lossy) mappings between them. This is useful, for example, to locate objects on a ship that also crosses the ocean. In this scenario, objects could be located relative to the ship or relative to a world coordinate system.

Advantages of the geometric model:

- Accuracy of information is preserved unless the mapping between coordinate systems is lossy.

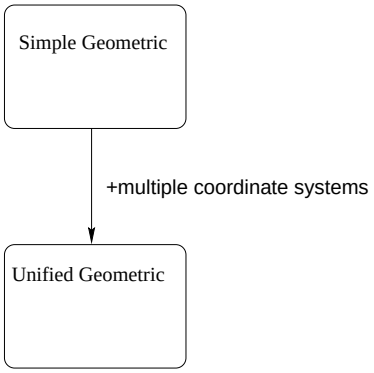


Figure 3.2: Classification of geometric models

- Coordinate-based spatial queries provide a very flexible means of retrieving location information.
- Sensors and applications only need to understand one of the reference coordinate systems in order to interact with the location service.
- Reference coordinate systems are typically reusable without customisation.

Disadvantages of geometric models:

- Coordinate data is weakly structured, which makes efficient design more difficult.
- All the information acquired needs to be translated to one of the reference co-ordinate systems. For example, an Active Badge network would need the coordinates for the coverage areas of all the sensors.
- Applications are burdened with unnecessary geometric data and computations.
- Management, especially access control, requires symbolic location.
- A separate location directory is required to map coordinates into data meaningful to applications and people.

3.1.3 Summary

Both models address different requirements for the management and processing of location data. In some cases, especially for closed systems with fixed requirements and provisions, one of the models can provide the appropriate solution. For example, an Active Badge network would typically be modelled using the symbolic approach. A single GPS tracking system maps well onto a geometric model. However, in the general case we believe that it is necessary to combine the two approaches. In particular, a general location service needs to accommodate both models of location information.

3.2 Symbolic models

In this section, we describe three models that represent symbolic location information: cell model, zone model, and domain model. These models are not mutually exclusive. A processing pipeline for location data could utilise all of them in different processing stages. These concepts were first proposed in [34].

Definition 1 *A* **located-object** *is an object whose location is of interest to the location service.*

The term located-object (first used by Schilit [59]) refers to all the real-world entities (people and hardware) that can be tracked by the location service.

Definition 2 *A* **symbolic location** *is a name referring to a well-defined geographical area which does not need to be constant over time.*

Symbolic location models differ in the way they construct and structure symbolic locations.

3.2.1 The cell model

In this model, assume that each location sensing system represents an object's location in terms of a well-defined geographical area (e.g., a room, a square on the map, an IR or RF cell, etc.). This area we call a *cell*. In the case of GPS, the cell's area is a circle defined by the sighting coordinates and the accuracy margins.

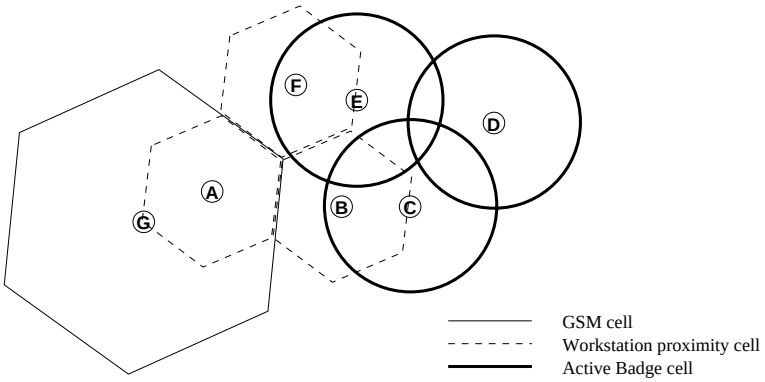


Figure 3.3: The geographical view of a simple cell space

Figure 3.3 gives an example of a cell space. There, we have seven cells labelled from 'A' to 'G'. Three overlapping sensor systems are involved: Active Badges, GSM, and the UNIX `ruser` service. A single location sighting can reference only one cell, even though the object could be in multiple cells at the same time.

Cells are the symbolic locations in this model. They are allowed to overlap because more than one sensor system can be included in the system. The inclusion relationship among cells is not represented in the cell model. Hence, this is a *simple symbolic model*.

The cell model preserves the accuracy of incoming information because it defines symbols for all locations referenced in sightings. However, it is awkward to use for spatial queries because of overlaps that it cannot represent. For example, it is impossible to enumerate all objects within a given cell in this model.

3.2.2 The zone model

In a cell space (such as shown by Figure 3.3), the intersection of the cells generates a set of mutually exclusive (i.e. non-overlapping) locations. We call these locations *zones*. Each zone is part of one or more cells.

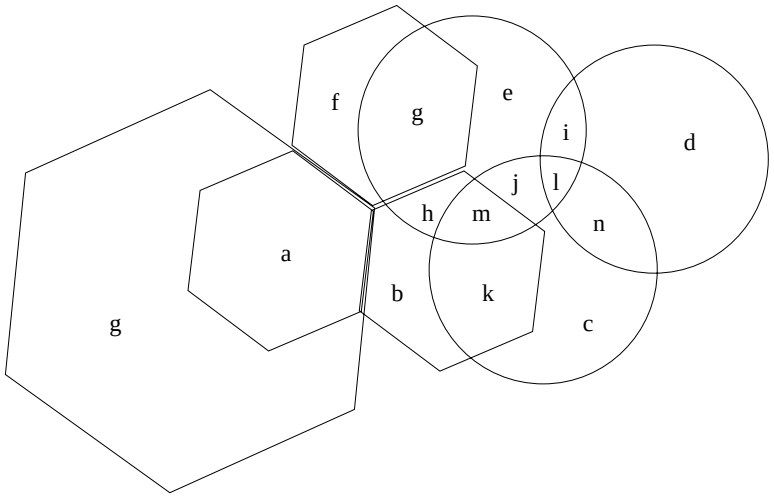


Figure 3.4: The geographical view of a simple zone space

Figure 3.4 shows a zone space derived from the cell space given in Figure 3.3. Now, the cells have been divided up into non-overlapping zones labelled from 'a' to 'n'. If an object is located in zone 'a', for example, it should be visible in cells 'A' *and* 'G' (cf. Figure 3.3).

In the zone model, we use zones rather than cells as symbolic locations. Since zones do not overlap, the zone model can be classified as an *exclusive symbolic model*. A single zone space can accommodate an arbitrary number of cells, the only prerequisite being knowledge of their respective overlaps. Thus different and overlapping cell spaces generated by different location sensing systems can be integrated.

Since zones do not overlap, a located-object can be in at most one zone at a time. Therefore, within a zone space, the movements of one object can be modelled by a single finite-state-machine. Hence, a zone space is a natural framework for persistent object tracking and movement prediction.

By defining the zones as the overlaps between different cells (see Figure 3.4), we achieve a potentially higher positioning accuracy. The zone space can also be partitioned into independent geographical coverage areas, and therefore zone space computations can be distributed.

However, the zone space has some shortcomings. Firstly, there is no notion of abstraction or multi-resolution processing. Secondly, the zone space for one located-object may

be entirely different from another's if both are visible to different location sensor systems.

3.2.3 The location domain model

A location domain is a symbolic location that can be ordered with respect to other location domains. Any set of location domains is partially ordered by the "contains" relationship. This ordering reflects the spatial inclusion of the underlying geographical areas. As in the cell model, domains are allowed to overlap.

An example is shown in Figure 3.5. The graph of the seven location domains forms a *lattice*. The 'College' domain includes, directly and indirectly, all the other domains. Also shown is a possible mapping from the zones of Figure 3.4 into the domain space.

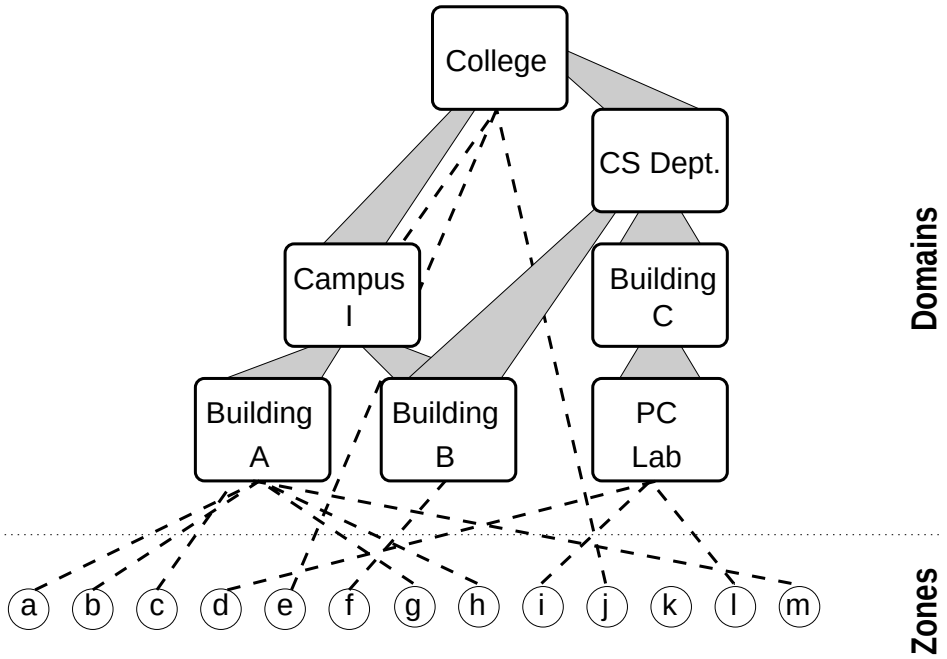


Figure 3.5: A partially ordered set of location domains

When employing this model, a location service typically uses a predefined set of domains to represent the locations of the located-objects. Located-objects join and leave domains as they move in geographical space. If a located-object is a member of a particular location domain, it must also be a member of all "parents" of this domain. To preserve strong consistency, changes in the domain membership should instantly propagate up the domain hierarchy. However, implementations may choose a weaker level of consistency.

An arbitrary set of location domains does not necessarily include a "root" location, that is, a location domain spatially containing all other domains. However, it is desirable to include such a domain in the domain space used by a location service. This root domain we shall call "anywhere".

In principle, a location domain can refer to a non-constant (i.e. mobile) geographical area. As a result, the position of the mobile domain within the inclusion ordering is variable over time. Also, changes in domain membership may be triggered both by movements of

the domain or the located-object. Supporting mobile domains carries an additional cost, hence it may not be implemented unless required.

Non-mobile objects can also be location domain members. Domain membership queries can thus accomplish basic location-dependent mapping functions (Example: find a telephone in this room).

The partial ordering of location domains is a very powerful mechanism, which can be exploited to gain scalability and manageability. Because they are sensor-independent and support multi-resolution processing, location domains provide a flexible framework both for client interaction and management operations. Later in this thesis we describe how the hierarchical nature of the location domains can be exploited to design scalable architectures and access control mechanisms.

Specification Since hierarchical location domains are a core theme of this thesis, we have specified them in the Z notation [52]. A brief introduction to Z is given in appendix B. By using this formalism, we hope to communicate our ideas concisely and without ambiguity. We have chosen the Z notation because it is well-suited for specifying state-based systems, and because automated checking tools are available. All the Z specifications in this thesis have been syntax-checked and type-checked using the Z/Eves tool [45]. A condensed specification is provided in appendix C.

For specifying the location domain model, the entities of interest are symbolic locations and located-objects.

$$[\mathbf{LOCATION}, \mathbf{OBJECT}]$$

We define an ordering relation for locations that reflects spatial inclusion. For an ordering relation, asymmetry and transitivity are essential properties. Asymmetry means, for example, that a *Building* containing a *Room* cannot at the same time be contained by that *Room*. Note that asymmetry implies irreflexivity, i.e. a location cannot be contained by itself. We have chosen to exclude reflexivity to keep the model simple. The location ordering must also be transitive. For example, a *Desk* in a *Room* in a *Building* must also be in the *Building*.

The following schema template specifies a generic irreflexive partial ordering, which we shall instantiate later for locations:

IrreflexivePartialOrder[**X**]

$$\_ < \_ : \mathbf{X} \leftrightarrow \mathbf{X}$$

$$\forall \mathbf{x}, \mathbf{y} : \mathbf{X} \bullet$$
$$\mathbf{x} < \mathbf{y} \Rightarrow \neg\, \mathbf{y} < \mathbf{x}$$
$$\forall \mathbf{x}, \mathbf{y}, \mathbf{z} : \mathbf{X} \bullet$$
$$\mathbf{x} < \mathbf{y} \land \mathbf{y} < \mathbf{z} \Rightarrow \mathbf{x} < \mathbf{z}$$

When instantiated for locations, this template forms the core of our specification of a location hierarchy. We specify a **LocationHierarchy** as consisting of an ordering

relationship between locations, a root location **anyLoc**, and the relation **conflicts**:

LocationHierarchy

$$\begin{array}{l}
\mathbf{IrreflexivePartialOrder}[\mathbf{LOCATION}] \\
\mathbf{anyLoc} : \mathbf{LOCATION} \\
\mathbf{conflicts} : \mathbf{LOCATION} \leftrightarrow \mathbf{LOCATION} \\
\hline
\forall l1, l2 : \mathbf{LOCATION} \bullet \\
\quad (l1, l2) \in \mathbf{conflicts} \Leftrightarrow \\
\quad\quad (\forall l3 : \mathbf{LOCATION} \bullet \\
\quad\quad\quad \neg ((l3 < l1 \lor l3 = l1) \land (l3 < l2 \lor l3 = l2))) \\
\forall l : \mathbf{LOCATION} \bullet \\
\quad l < \mathbf{anyLoc} \lor l = \mathbf{anyLoc}
\end{array}$$

In the above schema, **conflicts** is defined as a convenience relation. Two locations conflict if they do not overlap, i.e. if there is no third location contained by both. Further, we specify that **anyLoc** should contain all other locations, which intuitively corresponds to our idea of an all-inclusive root location.

For example, a simple location hierarchy could be constructed as follows:

$$\begin{array}{rcl}
\mathbf{LOCATION} & = & \{\mathbf{HuxleyBldg}, \mathbf{MathsDpt}, \mathbf{CompDpt}\} \\
_ < _ & = & \{(\mathbf{MathsDpt}, \mathbf{HuxleyBldg}), (\mathbf{CompDpt}, \mathbf{HuxleyBldg})\} \\
\mathbf{anyLoc} & = & \mathbf{HuxleyBldg} \\
\mathbf{conflicts} & = & \{(\mathbf{MathsDpt}, \mathbf{CompDpt}), (\mathbf{CompDpt}, \mathbf{MathsDpt})\}
\end{array}$$

In the above example, we consider a building which is shared by two administrative domains: the departments of mathematics and computing. The building serves as root location. The two sub-areas conflict with each other since they do not overlap.

The schema **SymbolicLocator** models the current locations of located-objects by associating objects with locations. An object can be in zero, one, or more locations at a time.

SymbolicLocator

$$\mathbf{locatedAt} : \mathbf{OBJECT} \leftrightarrow \mathbf{LOCATION}$$

Note that this specification allows objects to be in several non-overlapping locations at a time. We have not excluded this case in order to make the specification usable for components that do not ignore inconsistency.

3.3 A geometric model

Today GPS is the dominant outdoor positioning system. Hence location-awareness in many applications means processing of geographical coordinates. While some of those

applications could work with symbolic information if it were available, there are some that require a geometric view on location data. To address the requirements of geometric location processing, we propose a geometric location model, which can then be integrated into a hybrid semi-symbolic model.

Geometric location data is provided as a set of coordinates with respect to some reference co-ordinate system. For GPS, this is often longitude, latitude, and altitude in the `WGS84` reference system [13]. Since this in itself is an adequate mathematical model of geometric location, we only need to concern ourselves with defining symbolic location abstractions over the coordinate space.

Location areas (such as cities, buildings, or rooms) are represented by the coordinates of 2-D areas or 3-D volumes, in contrast to the purely name-based symbolic models.

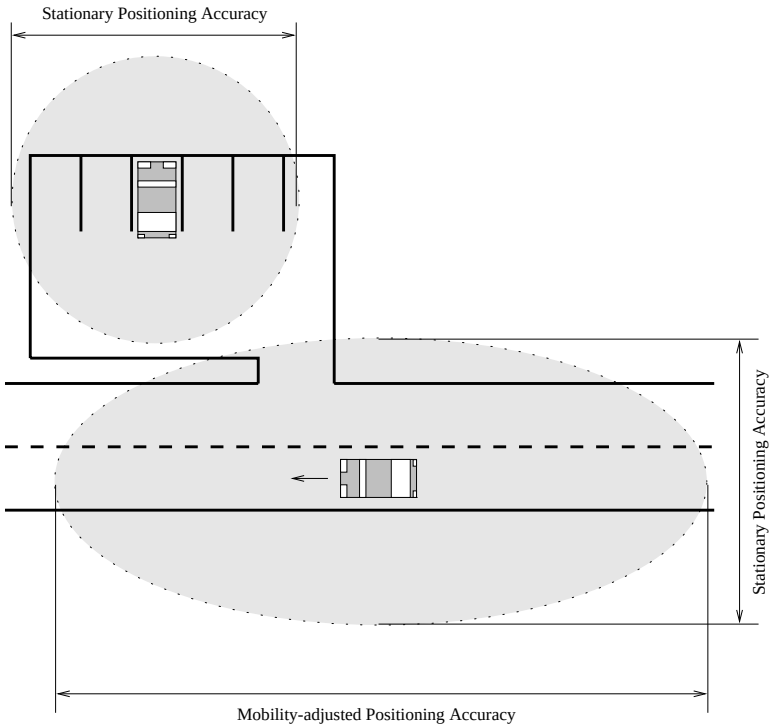


Figure 3.6: Location uncertainty areas for moving and stationary objects

With respect to located-objects, there are two distinct concepts to be modelled geometrically: position and layout. Position is ideally modelled by a point (i.e. a set of coordinates with respect to a reference coordinate system). However, no positioning or tracking system is error-free. So the located-object's position is best represented by an area of positioning uncertainty (Figure 3.6). The located-object's layout, which is especially important for large located-objects (e.g. a super-tanker), is modelled in the same way as are "normal" location areas. We shall not consider layout areas in greater details because we focus on the case where positioning uncertainty is significantly greater than the footprint of the located-object.

Spatial relationships between location areas can be established: inclusion, equiva-

lence, overlap, adjacency. These relationships can be translated into constraints over the coordinates representing the respective areas. It should be noted that tests for spatial relationships are non-trivial if the shapes of the areas involved are complex.

Specification Similar to the symbolic model, the entities of interest are locations and located-objects. Here, we model locations as areas. We do not include the notion of a point in our model because any point can be represented as a very small area. Hence, we declare a new sort of area object:

$$[\mathbf{AREA}]$$

The sort for located-objects has already been introduced in section 3.2.3.

The geometric location model specifies spatial relationships between areas. We consider overlap and strict inclusion as the primary relationships. The latter is asymmetric and transitive, which makes it a partial order. Additionally, geometric inclusion with equivalence is specified for convenience.

GeometricLocationModel

$$\begin{array}{l}
\mathbf{anyArea} : \mathbf{AREA} \\
\mathbf{contains} : \mathbf{AREA} \leftrightarrow \mathbf{AREA} \\
\mathbf{overlaps} : \mathbf{AREA} \leftrightarrow \mathbf{AREA} \\
\mathbf{containsEq} : \mathbf{AREA} \leftrightarrow \mathbf{AREA} \\
\hline
\forall\, \mathbf{a}1, \mathbf{a}2 : \mathbf{AREA} \bullet \\
\quad (\mathbf{a}1, \mathbf{a}2) \in \mathbf{contains} \Rightarrow (\mathbf{a}2, \mathbf{a}1) \notin \mathbf{contains} \\
\forall\, \mathbf{a}1, \mathbf{a}2, \mathbf{a}3 : \mathbf{AREA} \bullet \\
\quad (\mathbf{a}1, \mathbf{a}2) \in \mathbf{contains} \land (\mathbf{a}2, \mathbf{a}3) \in \mathbf{contains} \Rightarrow (\mathbf{a}1, \mathbf{a}3) \in \mathbf{contains} \\
\forall\, \mathbf{a} : \mathbf{AREA} \bullet \\
\quad (\mathbf{anyArea}, \mathbf{a}) \in \mathbf{containsEq} \\
\forall\, \mathbf{a}1, \mathbf{a}2 : \mathbf{AREA} \bullet \\
\quad (\mathbf{a}1, \mathbf{a}2) \in \mathbf{overlaps} \Leftrightarrow \\
\qquad (\exists\, \mathbf{a}3 : \mathbf{AREA} \bullet (\mathbf{a}1, \mathbf{a}3) \in \mathbf{contains} \land (\mathbf{a}2, \mathbf{a}3) \in \mathbf{contains}) \\
\forall\, \mathbf{a}1, \mathbf{a}2 : \mathbf{AREA} \bullet \\
\quad (\mathbf{a}1, \mathbf{a}2) \in \mathbf{containsEq} \Leftrightarrow (\mathbf{a}1, \mathbf{a}2) \in \mathbf{contains} \lor \mathbf{a}1 = \mathbf{a}2
\end{array}$$

It should be considered that in contrast to the symbolic locations, the number of possible geometric areas is infinite. Therefore the computation of the spatial relationships necessarily involves coordinate-based numerical computations.

The geometric locator associates located-objects with a geometric area, the positioning uncertainty area. We specify that each located-object has *one* geometric position.

GeometricLocator

$$\mathbf{position} : \mathbf{OBJECT} \rightarrow \mathbf{AREA}$$

The specification for the geometric locator is very similar to the symbolic locator. The main difference is that each located-object has only one geometric position but possibly a multitude of symbolic locations.

3.4 A combined model

In preceding sections, we have discussed symbolic and geometric location models. Applications require both types of model, sometimes even simultaneously. On the other hand, location sensors often support only one. Hence there is a strong case for an additional layer of indirection in order to decouple sensor representation from application representation. Such a layer needs to integrate both symbolic and geometric location information. Therefore, a combined model is needed which allows location data to be viewed either symbolically or geometrically (Figure 3.7).

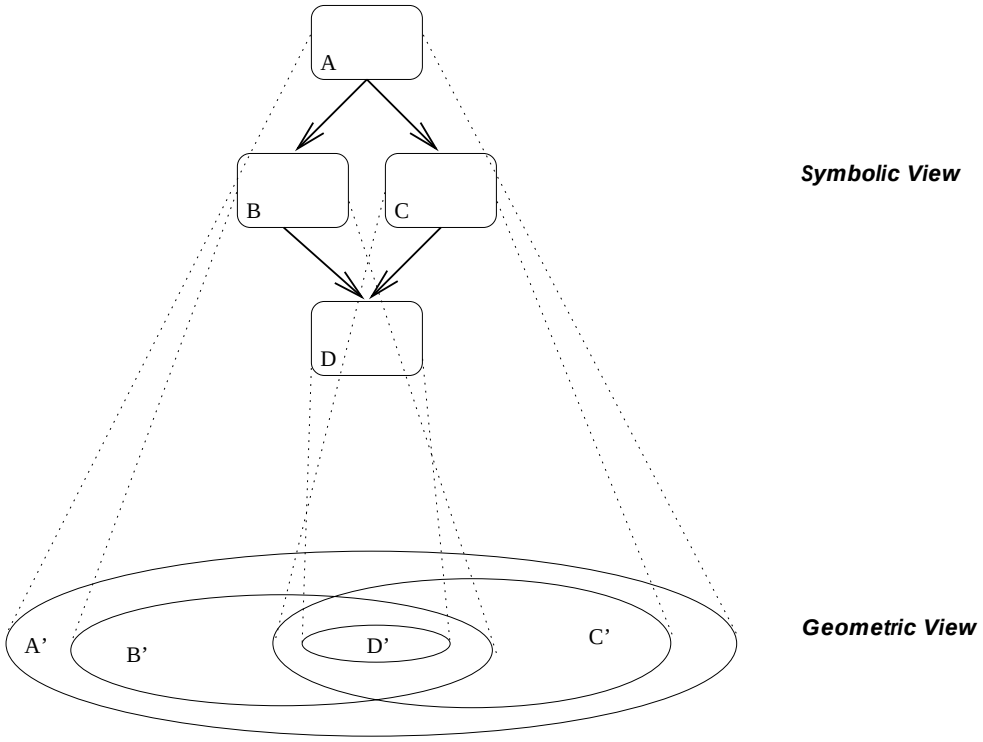


Figure 3.7: Symbolic and geometric views on location data

In order to combine the symbolic and geometric approaches, we need to identify corresponding entities in the two models and map them into a single entity in the combined model.

In the geometric model, a *located-object* corresponds to both an area of positioning uncertainty and a layout area. In the symbolic model, a located-object is an entity that can be a member of one or more location domains.

We primarily wish to track many small located-objects. Hence, we assume that the positioning uncertainty area is the primary piece of location information to be associated

with a located-object. Thus, an object's location is defined geometrically by the coordinates of its positioning uncertainty area, and symbolically by its membership in location domains.

Note that a located-object can be primitive (e.g. an Active Badge) or composite (e.g. a person wearing an Active Badge *and* a GPS receiver). In the scope of this work we shall assume that the composition of a located-object does not change (e.g. an Active Badge is worn all the time by its designated user).

In the combined model, the abstraction of a *location* corresponds to two distinct geometric meanings:

- A well-defined fixed area. Example: a room, a building, or a country.
- A large mobile object. This case covers also **zones of proximity** around mobile objects. Example: a big ship, an aeroplane, the danger zone around a hungry tiger.

Thus, a location is essentially a well-defined partition of space (e.g. 2-D area or 3-D volume). The definition of any such partition may change over time. However, for each point in time there is at most one such mapping from location to space.

Note: To keep things simple, we do not include time in our model. Hence, a single instance of the model is valid only for spans of time that do not include changes of locations' definitions.

Specification The specification of the new, semi-symbolic location model incorporates both the pure symbolic and the geometric specifications from the previous sections. Additionally, mappings are specified in both directions to relate the two views. The mapping **area** assigns to each location a geometric area. The function **leastMatchingLoc** finds the best matching defined location for a geometric area, while any geometric location is matched by **anyLoc**.

SemiSymbolicLocationHierarchy

$$
\begin{array}{l}
\textbf{LocationHierarchy} \\
\textbf{GeometricLocationModel} \\
\textbf{area} : \textbf{LOCATION} \rightarrow \textbf{AREA} \\
\textbf{leastMatchingLoc} : \textbf{AREA} \nrightarrow \textbf{LOCATION} \\
\hline
\textbf{area}(\textbf{anyLoc}) = \textbf{anyArea} \\
\textbf{leastMatchingLoc}(\textbf{anyArea}) = \textbf{anyLoc} \\
\forall \mathrm{l1}, \mathrm{l2} : \textbf{LOCATION} \bullet \\
\quad (\mathrm{l1} < \mathrm{l2}) \Leftrightarrow (\textbf{area}(\mathrm{l2}), \textbf{area}(\mathrm{l1})) \in \textbf{contains} \\
\forall \textbf{a} : \textbf{AREA}; \mathrm{l3} : \textbf{LOCATION} \bullet \\
\quad \textbf{leastMatchingLoc}(\textbf{a}) = \mathrm{l3} \Leftrightarrow (\textbf{area}(\mathrm{l3}), \textbf{a}) \in \textbf{containsEq} \wedge \\
\qquad (\forall \mathrm{l4} : \textbf{LOCATION} \bullet \\
\qquad\quad (\textbf{area}(\mathrm{l4}), \textbf{a}) \in \textbf{containsEq} \Rightarrow ((\mathrm{l3} = \mathrm{l4}) \vee (\mathrm{l3} < \mathrm{l4})))
\end{array}
$$

Finally, the specification of **SemiSymbolicLocator** specifies that the symbolic and

geometric positions of each object are known. Again, both the original symbolic and geometric specifications have been incorporated.

SemiSymbolicLocator

SymbolicLocator
GeometricLocator
SemiSymbolicLocationHierarchy

$\forall \mathbf{o} : \mathbf{OBJECT}; \mathrm{l} : \mathbf{LOCATION} \bullet$
$(\mathbf{o} \in \mathrm{dom}\, \mathbf{locatedAt}) \Rightarrow (\mathbf{o} \in \mathrm{dom}\, \mathbf{position})$
$\forall \mathbf{o} : \mathbf{OBJECT}; \mathrm{l} : \mathbf{LOCATION} \bullet$
$(\mathbf{o}, \mathrm{l}) \in \mathbf{locatedAt} \Rightarrow (\mathbf{area}(\mathrm{l}), \mathbf{position}(\mathbf{o})) \in \mathbf{containsEq}$
$\forall \mathbf{o} : \mathbf{OBJECT}; \mathrm{l1}, \mathrm{l2} : \mathbf{LOCATION} \bullet$
$((\mathbf{o} \in \mathrm{dom}\, \mathbf{position}) \wedge$
$(\mathbf{area}(\mathrm{l1}), \mathbf{position}(\mathbf{o})) \in \mathbf{containsEq} \wedge$
$(\mathbf{o}, \mathrm{l2}) \in \mathbf{locatedAt}) \Rightarrow ((\mathrm{l1} = \mathrm{l2}) \vee (\mathrm{l2} < \mathrm{l1}))$

With the combined model, we have proposed a general abstraction that is powerful enough to subsume both symbolic and geometric location processing. Hence, it provides a suitable data model to build an abstraction layer between location sensors and location-aware applications.

Implementation The database-centric location service described in appendix D is an implementation of the semi-symbolic hierarchical location model specified above. For performance reasons, we have chosen to implement the **position** and **area** functions as tables, while the **locatedAt** relationship is computed on-demand. As a result, the prototype maintains two pieces of state: an object table, which stores located-objects' geometric positions, and a location table, which maps location names into geometric areas.

3.5 A notation for location information

The location model described above specifies the semantic structure of our location model. Now it is necessary to introduce a notation that is a textual representation of the entities and relationships of our model. It is especially important to find a consistent and scalable naming scheme for locations.

Also, we aim to make location information amenable to formal reasoning using well-established formalisms such as first-order logic or temporal logic. For example, we propose to use first-order logic to specify logical constraints over location information. These constraints are a powerful tool for the specification of location-dependent behaviour and location-dependent queries.

In this section, we firstly describe a naming scheme for symbolic locations. Then we proceed to define basic and derived predicates over location information. We show how those predicates can be used to specify location-aware behaviour and queries.

3.5.1 Naming

It is necessary to name the objects of interest, that is *locations* and *located-objects*, in a consistent and intuitive manner. We shall focus on the former since the latter appears to be fairly well understood (see [49] for a discussion of naming). Examples of names for located-objects are the International Mobile User Numbers ([15]) or Internet email addresses [11].

A name space for locations needs to address to following issues:

- Location names must be hierarchical. This is necessary for scalability. Also, fixed locations are intuitively hierarchic.
- The design of the name space should intuitively reflect the physical location space. However, this must be a loose coupling since the name space should be less dynamic than the location space. In particular, movement of a mobile location should not necessitate a change in the name space.
- Individual locations have characteristics which are important for location-awareness: Locations can either be fixed or mobile, and their extent can be specified by either area geometry (e.g. "circle of radius 4") or an abstract concept (e.g. "Room"). While these distinctions should be recognisable in the notation, it is undesirable to reflect this in the semantic model.

Additionally, there are a number of pre-existing naming schemes for locations. These have the advantage of familiarity and maturity. Therefore, we would like to incorporate those schemes. We take into consideration the following approaches:

- Path names for fixed symbolic locations. This assumes a static acyclic graph with a root location. There may be more than one name for a location. Such a namespace is very similar to the Internet's DNS [46] or the naming in management domains [72]. For example, "/UK/London/Imperial/Huxley/336" identifies a particular room in Imperial College.
- Coordinate tuples to identify a geometric position. It is important to know the reference coordinate system. Example: "WGS84:(0.04W, 51.3N, 0)" identifies a particular point in the London area. The reference coordinate system is WGS84 (World Geodetic System 1984).
- Abstract locations. Natural language contains number of instantly recognisable and fairly unambiguous location abstractions, e.g. "room", "river", "town", or "country".
- Geometric shapes or volumes. The space covered by a location can be described mathematically by a geometric structure, such as a circle, a cube, or a rectangle,

In order to construct a namespace, it must be recognised that any concrete location contains two orthogonal dimensions: a geometric area, and the position of this area with respect to some reference point. Although the area component is sometimes implicit in the location's position, we think it is always desirable to explicitly state the area.

$$\text{location} \quad ::= \quad \langle \text{area} \rangle \text{ `@' } \langle \text{position} \rangle$$

If position is a fixed point, then the defined location is always static.[1] The area component is specified directly as a geometric area, or as an abstract area:

$$\text{area} \quad ::= \quad \langle \text{location concept} \rangle \mid \langle \text{geometric definition} \rangle$$

While the abstract area requires an additional level of indirection, it would be undesirable to specify all location with coordinates.

For the specification of an area's position, we need to distinguish between fixed positions and mobile positions. A mobile position is treated as a located-object.

$$\text{position} \quad ::= \quad \langle \text{located object} \rangle \mid \langle \text{fixed position} \rangle$$

For fixed positions, there remain two possibilities: specify a point with reference to some world coordinate system, or identify a "mount point" in a fixed structure.

$$\begin{array}{lll} \text{fixed position} & ::= & \langle \text{geometric position} \rangle \mid \langle \text{symbolic position} \rangle \\ \text{geometric position} & ::= & \langle \text{reference system} \rangle \text{ `:' } \langle \text{coordinates} \rangle \\ \text{symbolic position} & ::= & (\langle \text{symbolic position} \rangle \text{ `/' } \langle \text{label} \rangle) \mid \langle \text{well known position} \rangle \end{array}$$

The names for symbolic positions are hierarchical path names. Hence, they allow the construction of hierarchical location spaces. We assume that there is set of well known symbolic reference points, such as "New York City" or "Imperial College" which can serve as root locations. Note that a symbolic position usually implies a certain coverage area. By requiring the explicit specification of the area we force such hidden knowledge to be made explicit.

Below are some examples of locations specified with our notation:

`Room@Imperial/Huxley/529`	Room 529 in Huxley Building in Imperial College.
`Room@Jeff.Magee`	Room where Jeff Magee is currently located.
`<5m@Jeff.Magee`	Zone of proximity around Jeff Magee.
`<1m@WGS84:(0.04W, 51.3N, 0)`	Sphere around a spot somewhere in London

With this naming scheme we can concisely refer to locations of interest. Additionally, a resolution scheme is required in order to map names to locations. The resolution of location names depends on the nature and capabilities of the underlying location model. The resolution process requires that the location corresponding to the resolved name be identified. Such a location may be constructed if necessary.

In the previously described semi-symbolic location model, name resolution is implemented by two directory functions:

$$[\textbf{LOCNAME}, \textbf{OBJNAME}]$$

[1] We assume that areas are immutable.

NameResolver

$$\mathbf{resolveLoc} : \mathbf{LOCNAME} \pfun \mathbf{LOCATION}$$
$$\mathbf{resolveObj} : \mathbf{OBJNAME} \pfun \mathbf{OBJECT}$$

The **resolveLoc** mapping requires the **resolveObj** because locations can be attached to located-objects. The functions are partial because some names may not be resolvable. The resolution of location names comprises the following steps (but not necessarily in this order):

- Resolve references to located-objects. This requires the consultation of a location tracking service.
- Resolve aliases for symbolic positions. This requires an external directory which maps symbolic positions into a location reference known to the location service.
- Match position and area directly with the location hierarchy. For purely symbolic locations, this would result in a straight lookup of the location in question. If geometric information is involved, the corresponding location is found using a spatial index (such as an R-tree index [18]). The target location is found using both the area and position components of the name.

We feel that a further investigation of this area is necessary. However, this is outside the scope of this thesis.

3.5.2 Basic predicates

Since we have namespaces for locations and located-objects, we can define predicates of first-order logic representing common relationships within the location model. These are used to construct constraints over locations and located-objects. Primarily, we are interested in membership relationships between located-objects and locations, and in the spatial inclusion relationships among locations.

The predicates are defined over names and not directly over objects and locations. Thus, the previously defined semantic relationships (e.g. **contains** or **overlaps**) cannot be used directly.

Inclusion

$$\mathbf{NameResolver}$$
$$\mathbf{LocationHierarchy}$$
$$\mathbf{cont} : \mathbf{LOCNAME} \rel \mathbf{LOCNAME}$$

$$\forall \mathbf{n}1, \mathbf{n}2 : \mathbf{LOCNAME} \bullet$$
$$(\mathbf{n}1, \mathbf{n}2) \in \mathbf{cont} \Leftrightarrow \mathbf{resolveLoc}(\mathbf{n}1) \leq \mathbf{resolveLoc}(\mathbf{n}2)$$

Position

SymbolicLocator
NameResolver
$\mathbf{loc} : \mathbf{OBJNAME} \leftrightarrow \mathbf{LOCNAME}$

$\forall \mathbf{on} : \mathbf{OBJNAME}; \mathbf{ln} : \mathbf{LOCNAME} \bullet$
$(\mathbf{on}, \mathbf{ln}) \in \mathbf{loc} \Leftrightarrow (\mathbf{resolveObj}(\mathbf{on}), \mathbf{resolveLoc}(\mathbf{ln})) \in \mathbf{locatedAt}$

With the help of the predicates **cont** and **loc**, simple constraints about locations and located-objects can be specified. Here are some examples:

`loc(Fred,Room@Imperial/Huxley/429)`	Fred is located in 429 in Huxley
`loc(Joe,<4m@Fred)`	Joe is less than 4m from Fred
`loc(Fred,Room@Joe)`	Fred is in the same room as Joe
`cont(Room@Fred,Room@Joe)`	Fred is in the same room as Joe

Further examples can be found further below in section 3.5.5.

3.5.3 Collocations

An interesting abstraction is the notion of *collocation*. Informally, a collocation exists between located-objects that are spatially close to one another. The two located-objects are then said to be *collocated*. In one sense, collocation can be seen as the most basic symbolic location information. However, collocation is a relative notion. Therefore, it is not suitable as the primary concept in our model.

Formally speaking, collocation can be represented as a ternary relationship between two located-objects and a location pattern (which maps to a dynamic set of locations). In effect, collocation is a predicate that can hold between to objects with respect to a certain location pattern. It therefore lends itself quite naturally to describing conditions or assertions over location data.

$[\mathbf{LOCPATTERN}]$

A matching function maps patterns into a set of location names.

LocationPattern

$\mathbf{match} : \mathbf{LOCPATTERN} \rightarrow \mathbb{P}\,\mathbf{LOCNAME}$

Collocation

LocationPattern
Position
$\mathbf{colloc} : \mathbb{P}(\mathbf{OBJNAME} \times \mathbf{OBJNAME} \times \mathbf{LOCPATTERN})$

$\forall \mathbf{o1}, \mathbf{o2} : \mathbf{OBJNAME}; \mathbf{p} : \mathbf{LOCPATTERN} \bullet$
$(\mathbf{o1}, \mathbf{o2}, \mathbf{p}) \in \mathbf{colloc} \Leftrightarrow$
$(\exists \mathbf{ln} : \mathbf{LOCNAME} \bullet$
$(\mathbf{ln} \in \mathbf{match}(\mathbf{p})) \wedge ((\mathbf{o1}, \mathbf{ln}) \in \mathbf{loc}) \wedge ((\mathbf{o2}, \mathbf{ln}) \in \mathbf{loc}))$

Thus, two located-objects are collocated with respect to a particular pattern if, and only if, there is at least one common location which matches this pattern.

This notion of collocation is based purely on the syntactic matching of location names. The matching process can be optimised by using only locations corresponding to a zone of proximity if there are multiple matches (see examples).

Collocation is a powerful abstraction when it comes to specifying constraints. Here are some examples:

- `colloc(Joe,Fred,Room@429:Huxley)`. Joe and Fred are in Room 429 in Huxley.
- `colloc(Joe,Fred,<4m)`. Joe and Fred are less than 4m apart from each other. This can be rewritten since
  `colloc(Joe,Fred,<4m) iff loc(Joe,<4m@Fred) or loc(Fred,<4m@Joe)`.
- `colloc(Joe,Fred,Room)`. Joe and Fred are in the same room. Again, this can be optimised using
  `colloc(Joe,Fred,Room) iff loc(Joe,Room@Fred) or loc(Fred,Room@Joe)`

These examples show how the `colloc` predicate can be used to define constraints over the locations of located-objects. Such constraints can be readily employed in event conditions or location-dependent database queries.

3.5.4 Distance

So far in this chapter we have ignored all spatial relationships but inclusion (overlaps were derived from inclusion). While this allows an accurate static model of how located-objects relate to locations, it makes queries of the type "Find the nearest ..." difficult to answer. To solve this problem, we need to include a notion of *distance* into our location model.

[**DISTANCE**]

We do not specify how distance should be represented by the location model, but we do say that distance values should at least be partially ordered.

Distance

$$\textbf{IrreflexivePartialOrder}[\textbf{DISTANCE}]$$
$$\textbf{dist} : \textbf{OBJECT} \times \textbf{OBJECT} \rightarrow \textbf{DISTANCE}$$
$$\_ \leq \_ : \textbf{DISTANCE} \leftrightarrow \textbf{DISTANCE}$$

$$\forall \textbf{o}1, \textbf{o}2 : \textbf{OBJECT} \bullet$$
$$\quad \textbf{dist}(\textbf{o}1, \textbf{o}1) = \textbf{dist}(\textbf{o}2, \textbf{o}2)$$
$$\forall \textbf{o}1, \textbf{o}2 : \textbf{OBJECT} \bullet$$
$$\quad (\neg\, \textbf{o}1 = \textbf{o}2) \Rightarrow (\textbf{dist}(\textbf{o}1, \textbf{o}1) < \textbf{dist}(\textbf{o}1, \textbf{o}2))$$
$$\forall \textbf{d}1, \textbf{d}2 : \textbf{DISTANCE} \bullet$$
$$\quad (\textbf{d}1 \leq \textbf{d}2) \Leftrightarrow (\textbf{d}1 = \textbf{d}2 \lor \textbf{d}1 < \textbf{d}2)$$

Distance is directional, and it also depends on certain assumptions about the mobility of a located-object. The distance from **A** to **B** when going by car will be different from the distance by plane.

DistanceNamed

$$\textbf{Distance}$$
$$\textbf{NameResolver}$$
$$\textbf{dists} : (\textbf{OBJNAME} \times \textbf{OBJNAME}) \nrightarrow \textbf{DISTANCE}$$

$$\forall \textbf{o}1, \textbf{o}2 : \textbf{OBJNAME} \bullet$$
$$\quad \textbf{dists}(\textbf{o}1, \textbf{o}2) = \textbf{dist}(\textbf{resolveObj}(\textbf{o}1), \textbf{resolveObj}(\textbf{o}2))$$

Ideally, we would like to define distance on top of the previously defined location hierarchy. As it turns out, the notions of distance between objects and geographical inclusion between areas are completely orthogonal. Theoretically, the distance between two objects does not have any impact on the likelihood of the two being in the same location area. Only if the maximum possible distance between two objects in a given area is known does such a dependency exist. Therefore, we conclude that distance, along with geometric inclusion, is a basic and *axiomatic* notion of the location model.

Having the defined the concept of distance, we can now specify the notion of "nearest" (which is of great importance for location-awareness).

Nearest

$$\textbf{Distance}$$
$$\textbf{nearest} : \textbf{OBJECT} \times \mathbb{P}\,\textbf{OBJECT} \rightarrow \mathbb{P}\,\textbf{OBJECT}$$

$$\forall \textbf{o}1 : \textbf{OBJECT}; \textbf{O} : \mathbb{P}\,\textbf{OBJECT} \bullet$$
$$\quad \textbf{nearest}(\textbf{o}1, \textbf{O}) = \{\textbf{o}2 : \textbf{OBJECT} \mid$$
$$\qquad \textbf{o}2 \in \textbf{O} \land (\forall \textbf{o}3 : \textbf{OBJECT} \bullet$$
$$\qquad\quad (\textbf{o}3 \in \textbf{O} \Rightarrow (\textbf{dist}(\textbf{o}1, \textbf{o}2) \leq \textbf{dist}(\textbf{o}1, \textbf{o}3))))\}$$

This function yields a set of objects because multiple objects may have the same distance, and because the set of input objects may be empty.

A similar function needs to be specified over location names:

NearestNamed

$$\begin{array}{l} \textbf{Nearest} \\ \textbf{NameResolver} \\ \textbf{nearests} : (\textbf{OBJNAME} \times \mathbb{P}\,\textbf{OBJECT}) \nrightarrow \mathbb{P}\,\textbf{OBJECT} \\ \hline \forall\, \textbf{on} : \textbf{OBJNAME};\ \textbf{O} : \mathbb{P}\,\textbf{OBJECT} \bullet \\ \quad \textbf{nearests}(\textbf{on}, \textbf{O}) = \textbf{nearest}(\textbf{resolveObj}(\textbf{on}), \textbf{O}) \end{array}$$

Collocation versus Distance Collocation and Distance are different concepts with the common purpose of modelling spatial proximity between located-objects. In fact, we can construct locations as zones of proximity around located-objects to capture a binary notion of distance. However, collocation is not an appropriate abstraction when trying to find the located object with the minimum distance. On the other hand, distance is not very helpful if we are interested in a more abstract notion of distance, such as "in the same room" or "on the same street". Then, collocation is the more powerful abstraction.

3.5.5 Examples of use of locations, collocations, and distance

To conclude the section about our notation for location information, we have assembled some examples showing the application of constraints in event conditions, location-dependent queries, and for location-dependent access control.

Event Conditions When an event condition becomes true, the event is triggered. This principle is used, for example, by Schilit's context-triggered actions [58].

- `when colloc(Jeff Magee, Jeff Kramer, Room) and colloc(Jeff Magee, Morris Sloman, Room) => start the meeting`. Collocation is not necessarily transitive, but `colloc` is.
- `when loc(x, <3m@MyCar) => set off car alarm`. When somebody is near my car, set off the car alarm.
- `when loc(MyChild, >20@Me) => raise alarm` . When my child is more than 20 metres away from me raise an alarm.

Queries SQL with object-oriented extensions, such as used by Illustra [25], lends itself naturally to construct database queries with location information.

- Query "Find the nearest restaurant".
  `SELECT Name FROM Restaurants ORDER BY dists(Me,Name) ASC` (procedurally) or, `return nearests(Me, SELECT Name FROM Restaurants)` (declaratively)
- Query "Find a restaurant within 1 km".
  `SELECT Name,Address FROM restaurants WHERE dists(Me,Name)` $\leq$ `1km`

location	::=	⟨area⟩ '@' ⟨position⟩
area	::=	⟨location concept⟩ | ⟨geometric definition⟩
fixed position	::=	⟨geometric position⟩ | ⟨symbolic position⟩
geometric position	::=	⟨reference system⟩ ':' ⟨coordinates⟩
symbolic position	::=	(⟨symbolic position⟩ '/' ⟨label⟩) | ⟨well known position⟩
location predicate	:	loc(⟨object⟩,⟨location⟩)
collocation predicate	:	colloc(⟨object⟩,⟨object⟩,⟨pattern expression⟩)

Table 3.1: Summary of location notation

- Query "Who is with Jeff Magee in a room?".
  `SELECT Name FROM Persons WHERE colloc(Name, Jeff Magee, Room)`

Access control Recently, there has been interest in location-dependent access control. The idea is to modify a principal's access rights according to his or her physical location. Equally, it is important to prevent unauthorised access to location information.

- `A+: SOURCE` { $\lambda \mathbf{x}.\mathbf{loc}(\mathbf{x}, \mathbf{Room})$ } `TARGET`
  i.e. Members of `SOURCE` may evaluate the **loc**() predicate over **Room** and members of `TARGET`.

- `A+: SOURCE` { $\lambda \mathbf{l}.\mathbf{colloc}(\mathbf{x}, \mathbf{y}, \mathbf{l})$ } `TARGET`
  i.e. Members of `SOURCE` may evaluate the collocation predicate with respect to any pattern from `TARGET`.

- `A+: SOURCE { access } TARGET WHEN loc(SOURCE, France)`
  i.e. Members of `TARGET` are only accessible from within `France`.

The policy syntax is based on [44], except for the lambda notation, introduced to clarify which of the constraint's parameters is the policy's target.

3.6 Chapter Summary

In this chapter, we have examined location models to support location-aware applications and services. We have clarified the distinction between geometric and symbolic location models. From an analysis of existing location-aware applications and location sensors we have concludeds that both representations need to be accommodated by a general location service.

Hence, we have defined and specified geometric and symbolic models with a view to integrate them. The result of the integration is a hybrid, semi-symbolic model that decouples application representation from the sensor representation of location information. The hybrid model can accept location data in both forms, and all data in the model can be viewed from both perspectives. This functionality is supported by a prototype

implementation (cf. appendix D). The hybrid model is hierarchical and thus allows for multi-resolution processing of location data. This feature is essential for building a scalable and manageable location service.

In order to facilitate name-based access we have defined a scalable naming scheme for locations, which is summarised in table 3.1. The scheme accommodates both symbolic and geometric location data. Also, mobile locations are supported. An algorithm to resolve such names has been outlined.

Reasoning about location data is further facilitated by predicates exposing the relationships among locations and located-objects. We have discussed the application of such predicates in constraints for event conditions, database queries, and access control conditions.

Chapter 4

Service Models

The previous chapters have described how locations can be modelled, and how location information can be represented. In this chapter, we exploit those models by proposing a general location service based on a hierarchical location model. We start by specifying the service functions for both symbolic and geometric location information. Then, we discuss how the components of the service functions map to concrete location service architectures. Finally, we outline an architecture for a global federated location service.

What is a location service?

The notion of a *service* is widely used in the fields of operating systems and distributed computing. However, it is useful to clarify at this point what is meant in this context by a service.

Definition 3 *A* **service** *is a shared object encapsulating shared resources.*

Definition 4 *A* **service access point** *is an instance of an interface to a service.*

To define a location service, we need to identify the shared resources encapsulated by it. These resources consist of:

- Positioning and tracking systems (Hardware), e.g. an Active Badge system.
- Historical location data (Dynamic data), e.g. the last known locations of located-objects.
- Geographical information (Static data), e.g. digital maps.
- Value-adding services (Service logic), e.g. event dissemination, access control, movement prediction.

A location service is a shared object that integrates those components in order to provide clients with a high-level interface to obtain location information. Location-aware applications are a category of such clients.

However, we should not base a definition on the above enumeration of encapsulated resources alone. What really matters is the pragmatic aspect: *What does it do?*

Definition 5 *A* **location service** *is a shared object that provides information about the physical location of located-objects.*

We do not specify the architectural framework nor a single set of functions for such a service. Both need to be chosen for a concrete *type* of location service (such as general location service, mobile telephone subscriber tracking, etc.) to address the balance of requirements for the target system.

We aim to provide a *toolkit* of functionality and architectural approaches to build different types of location services, and more generally, to make applications location-aware. So while our work primarily views a location service as a shared resource on the network, a stand-alone application (e.g. a GPS hand-held receiver) is seen as a degenerate case of a system with a location service.

4.1 Functional requirements and taxonomy

The definition given above leaves open many aspects of the design of a concrete location service. In the following, we examine the most important of those choices and their motivating requirements. Thus, we define a *design space* [33] for location services by identifying its dimensions, both functional and structural.

4.1.1 Location model

As discussed in some length in chapter 3, there are two main types of location model: geometric and symbolic. The choice of location model depends directly on the format supported by the sensors, and the formats required of the interface of the service. While the internal location model does not need to coincide with the internal format or formats, it makes sense to use an internal format that is easily translated to external formats. For example, if all inputs and outputs are geometric, it would most probably be a bad choice to employ symbolic locations internally.

The location model can also include knowledge of spatial relations between locations, especially inclusion and distance. Those relationships determine what kinds of queries are possible. For example, a "nearest" query necessitates a notion of *distance* between locations or objects. Similarly, multi-resolution tracking requires knowledge of the spatial *inclusion* relationship.

In some cases, the design phase for a location service not only establishes the location model to be used, but also defines types of locations and their characteristics (e.g. streets, highways, bridges in the case of a vehicle-tracking service). This results in very special-purpose location services, which can then be optimised for a certain operating environment. Such optimisation often includes static partitioning and distribution of the location information. Hence, the choice of the location model is often intertwined with the architecture, and especially with the distribution model of the target system.

Note: At this time, there does not seem to be a location service with a freely definable set of locations, perhaps with dynamically adapting (possibly self-organising) architecture. This is an avenue for future research.

4.1.2 Single vs. Multiple choices

The scope of the location service is narrowed by restricting certain types of entities to be singletons. Restrictions of this kind often reduce architectural complexity.

Tracked objects. With only one tracked object the location service degenerates to a personal tracking system. An example is a hand-held GPS receiver.

Monitored locations. This is the surveillance approach ("Who is in the building?"). However, in practice such systems tend to differentiate between more than one location. We believe that restricting a location service to a single location is not very useful.

Sensors. This can either mean one of the two cases discussed above, or we have a very capable sensor, possibly an integrated multi-sensor system appearing to be a singleton. For example, we can treat a closed location tracking system (such as an Active Badge system) as a single sensor.

Clients. Although a service by definition is not restricted to a single client, we wish to include the case where the location service is dedicated to serving a single instance of a location-aware application (such as a traffic-monitoring centre).

We concentrate on the general case: multiple instances of objects, locations, sensors, and clients.

4.1.3 Security

Open systems, by definition, are vulnerable to attacks on secrecy, availability, and integrity of information. In the context of a location service, two aspects are especially important:

Privacy Improper disclosure of people's locations infringes their privacy.

Trustworthiness Applications and their users may base decisions on location data. Hence the service must be trusted either to deliver correct and complete information, or to flag possible deficiencies.

Both issues apply also to other kinds of personal data, but location data is different in a number of ways. Firstly, the data itself is highly dynamic. Secondly, the need for privacy is variable, depending, for example, on the person's current environment. Thirdly, there are other paths for the disclosure of location information (e.g. through personal observation). An unnecessarily secretive location service will limit functionality without a gain in security. Finally, the trustworthiness of location data is challenging because most of the information is gathered automatically from sensors that are vulnerable to deception.

These challenges can be addressed in a number of ways by the architecture and design of the system. Some systems may rely on auditing measures (such as revealing the identity of the person querying somebody's location), others will prefer enforced access control.

There appears to be a trade-off between the involvement of the service itself in the security measures (undesirable) and the granularity of those measures (fine-grained control is desirable).

4.1.4 Correctness and completeness of information

Ideally, a location service should provide correct and complete location information. However, both qualities are difficult to achieve because of the physical characteristics of positioning and tracking systems.

Many applications survive without perfect information. The Internet's current domain naming service (DNS [46]) sometimes fails to resolve a name (incompleteness), or the resolution yields the wrong IP address (incorrectness). Hence, applications have no choice but to tolerate these deficiencies. We believe a similar approach is viable and necessary for large-scale location service implementations. This view is also supported by the end-to-end argument [57].

Nevertheless, higher quality information will be needed in some cases. This may require special location sensor systems that cannot be fooled without major effort. Further, the application needs to be made aware of the quality (trustworthiness) of the data it receives, by tagging either data items or service instances as a whole.

4.2 Functional decomposition

We have defined the location service in terms of its minimum functionality (definition 5), which is basically a tracking function. In practice, this tracking function will require some additional knowledge: a location directory and a spatial relationship database. Further, value-added services, such as tracing and prediction may be required. Hence, a location service must be seen as a set of services, configured by local requirements.

We have identified a tool-kit of functionality used to design and implement a location service. Each element corresponds to an 'encapsulated resource' (page 67).

- **Tracking Function** (Historical location data). This function provides the last observed positions of located objects. It can be subdivided into an *object tracking function* which keeps track of selected located-objects, and a *location monitoring function*, which monitors a selected location for located-objects. This function is also responsible for providing sighting histories if required.

- **Directory Function** (Geographical information). The directory function is responsible for translating between different representations of location information. It includes a *symbolisation function*, a *desymbolisation function*, and a *coordinate translation function*. We have already specified a simple directory function with **asArea** in § 3.3 (page 53), and **area/leastMatchingLoc** in § 3.4 (page 55).

- **Spatial Relationship Function** (Geographical information). This provides information about the spatial relationship between locations. Knowledge of spatial relationship is important both for acquisition and dissemination of location information. Digital maps and movement planes are examples of this category. The spatial relationships of interest are *inclusion* (specified in § 3.2.3, page 51), and *distance* (specified in § 3.5.4, page 61).

- **Acquisition Function** (Positioning and tracking system). This is responsible for gathering and pre-processing sensor data. It includes *reception function*, *abstraction function*, and *fusion function*. It is discussed in detail in chapter 5.

- **Trace Function** (Historical location data). The movement trace function reconstructs continuous movement models from the available discontinuous position measurements. Includes *history function*, and *prediction function*. We describe this function in chapter 6

- **Dissemination Function** (Value-adding services). This function is responsible for making the data provided by the other function accessible to clients. An important feature is asynchronous information, which may be provided either by events or event-triggered standing queries. The dissemination function consists of *query function*, an *event function*, and a *standing query function*.

- **Access Control Function** (Value-adding services). Access control prevents unauthorised disclosure of location information as well as unauthorised data acquisition. It consists of *security control* and *update control*.

The dependencies between these functions are shown in Figure 4.1. In the figure, a module depends on all modules beneath it. That is, a module can be used only if all supporting modules are present, too. However, it may be possible for module to work with a reduced functionality when non-essential supporting modules are missing. For example the Tracing function could work in reduced form (i.e. without interpolation or prediction) if the Spatial Relationship function is unavailable. Similarly, the dissemination function would work with only the tracking and directory functions supporting it.

4.3 Specification of the core functionality

The core function of the location service is tracking. Without the tracking function, we would have a geographical information system (GIS) at best. Also, tracking potentially involves a large amount of dynamic state and is therefore a key area of the architecture.

For these reasons, and in order to have a formal basis for the following chapters, we specify the tracking function using the Z-notation [52]. For the purposes of this specification, we shall consider a service that provides the object tracking function and the location monitoring function. This functionality is supported by a directory function and a spatial relationship function as specified in chapter 3. Further, we assume dissemination by a query function. The acquisition function is modelled by simple updates.

Our approach is valid for both symbolic and geometric models. We have specified tracking functions for symbolic, geometric, and hybrid models (see appendix C). Here, we focus on the symbolic case, which we believe to be the most important.

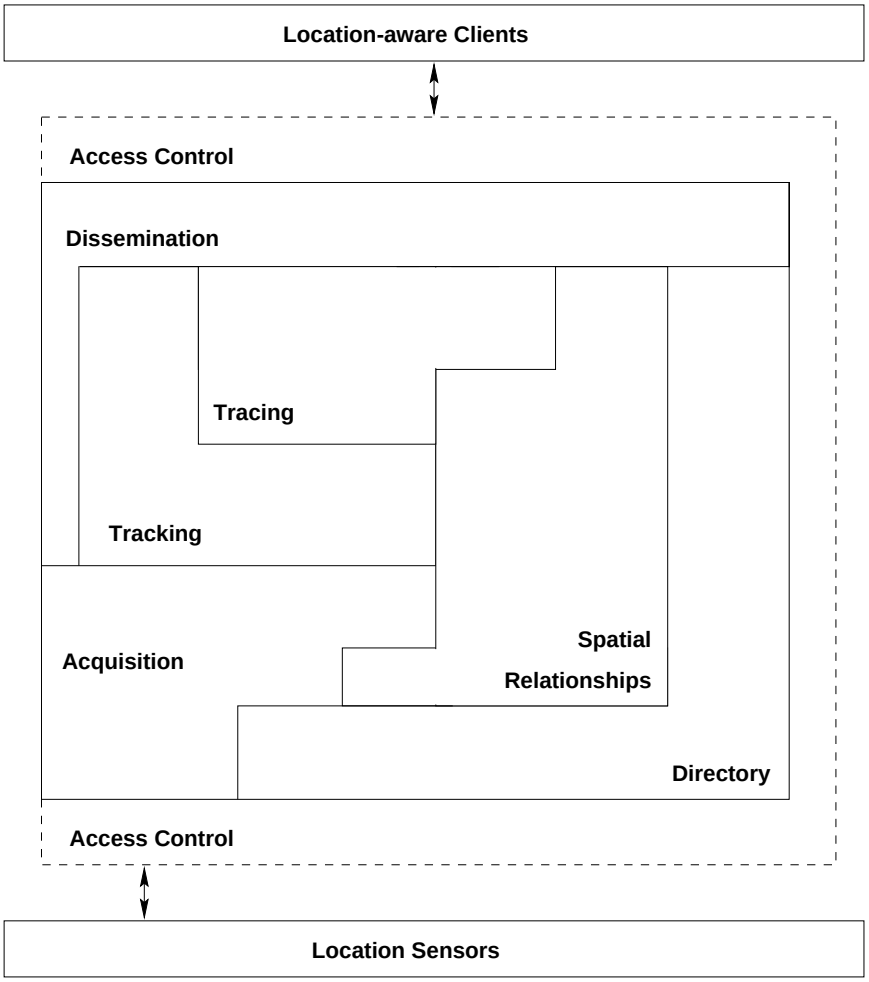


Figure 4.1: Layers of a location service

SymbolicLocationService

LocationHierarchy
SymbolicLocator

$\forall \mathbf{o} : \mathbf{OBJECT}; \mathrm{l1}, \mathrm{l2} : \mathbf{LOCATION} \bullet$
$\quad ((\mathbf{o}, \mathrm{l1}) \in \mathbf{locatedAt} \land (\mathbf{o}, \mathrm{l2}) \in \mathbf{locatedAt}) \Rightarrow (\mathrm{l1}, \mathrm{l2}) \notin \mathbf{conflicts}$

This schema states that the service uses a **LocationHierarchy** (cf. § 3.2.3, page 51) as the spatial relationship service. The **SymbolicLocator** (cf. § 3.2.3, page 51) provides an object tracking function. As a constraint, an object can only be in non-conflicting (that is, overlapping) locations at the same time.

SObjectQuery

Ξ**SymbolicLocationService**
target? : **OBJECT**
result! : $\mathbb{P}$ **LOCATION**

---

result! $= \{\mathbf{x} : \mathbf{LOCATION} | (\mathbf{target}?, \mathbf{x}) \in \mathbf{locatedAt}\}$

An object query returns the set of currently known locations for a given object. In theory, it could return also all the sublocations of each of the locations (see **SObjectQueryComplete**, appendix C). Since the location hierarchy is typically static, the simpler query should be the default behaviour. This is an instance of the end-to-end argument [57].

A location query returns all the objects at a given location, not including sub-locations:

SLocationQuery

Ξ**SymbolicLocationService**
target? : **LOCATION**
result! : $\mathbb{P}$ **OBJECT**

---

result! $= \{\mathbf{x} : \mathbf{OBJECT} | (\mathbf{x}, \mathbf{target}?) \in \mathbf{locatedAt}\}$

Similarly to schema **SObjectQueryComplete**, we have additionally specified an iterative version, **SObjectQueryComplete** (appendix C) which cycles through all the target's sub-locations. We prefer **SLocationQuery** because it is more likely to yield a scalable implementation. The iterative, complete versions of both queries may be implemented as an option.

Location updates are submitted by the acquisition function. We model updates so that an update overwrites all previous information in the locator:

SUpdate

Ξ**LocationHierarchy**
Δ**SymbolicLocator**
l? : **LOCATION**
o? : **OBJECT**

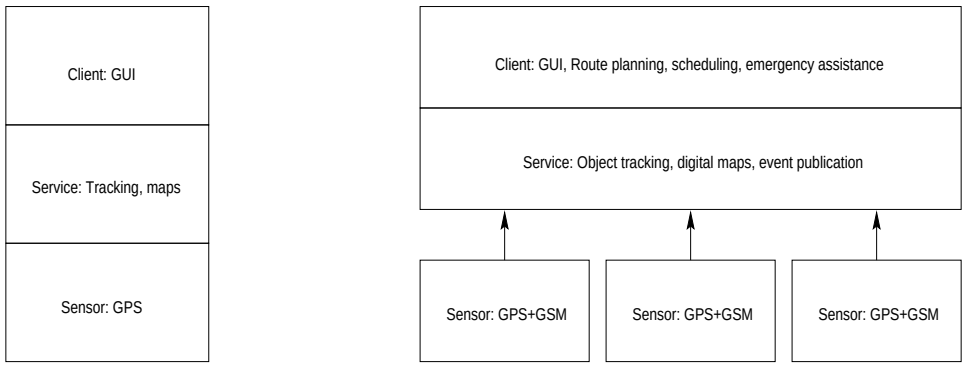
$$\textbf{locatedAt}' = (\{\textbf{o}?\} \lhd\!\!\!\!- \textbf{locatedAt}) \cup \{(\textbf{o}?, \text{l}?)\}$$

There are two potential pitfalls with this specification of an update. Firstly, a stateless service would not need updates. Hence, the update function should be seen as optional. Secondly, less precise updates will overwrite previous accurate updates. (A mobility model is required in order to detect this. Mobility models are functionally a part of the trace function. Chapter 6 offers a more detailed discussion of this area.)

Implementation The database-centric location service described in appendix D implements the above specification, along with queries and updates over geometric location information. However, the prototype's algorithmic structure differs from the specification because the **locatedAt** relationship is computed on-demand rather than for each update. This is permissible since the external behaviour matches the specification.

4.4 Architectural considerations

System model Location-aware systems are composed of location sensors, a location processing layer, and location-aware application (Figure 4.2). Focusing on distributed location-aware systems, we can further subdivide the location processing layer into a shared location service, as well as application and sensor agents (Figure 4.3). This structure is adopted from distributed directory services (see [19], page 823).



(a) hand-held GPS (b) vehicle-tracking system

Figure 4.2: Location-aware systems

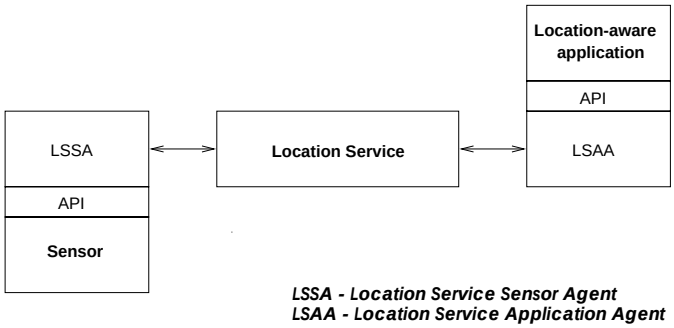


Figure 4.3: Service model: external interaction

Internally, the location service consists of two logical entities: location register (LR) and object register (OR) (Figure 4.4). The LR maintains location-specific state and performs location-centric processing. The OR maintains state information for located-objects and performs processing related to located-objects. Note that both LR and OR can be distributed. Further, both LR and OR can be implemented by the same facility, or be entirely stateless.

We model the register entities from the above discussion in terms of agent and database (Figure 4.5). By definition, a register contains persistent state. Entities that merely route and process information we call *processors*. These can be thought of as a register without database.

4.4.1 State

Some of the function of the location service, such as acquisition, tracking, tracing, and dissemination may contain *state*. For example, the last known position of each located-object is often persistently stored by a location service to make location information available when position measurements are impossible.

Which information is stored rather than computed on demand depends on the desired synchronisation patterns and on performance considerations. The same functional design can often be mapped to architectures that are very different from one another as far as the amount of state and its distribution are concerned.

For this discussion, we consider the state of the location service as a set of associations between locations and located-objects as specified in schema **SymbolicLocator** (page 51).

Stateless design. All queries ultimately trigger actual location measurements, for example a radio triangulation. This approach is appropriate if the service is used only infrequently. An example is the location-dependent emergency call feature for mobile telephones, called "wireless E-911" in the USA (see [85], page 259).

Maximum state. All observable location changes trigger state updates. Such a solution would be adopted if on-demand measurements are unavailable, or if the volume and frequency of queries outweighs updates. Here, a useful metric is the query-to-update ratio, also known as call-to-mobility ratio [28].

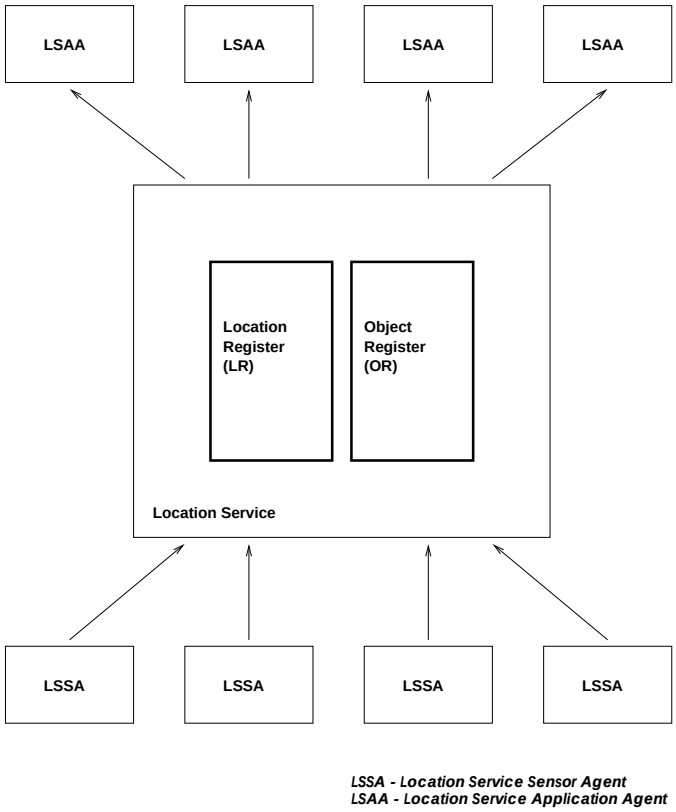


Figure 4.4: Logical architecture

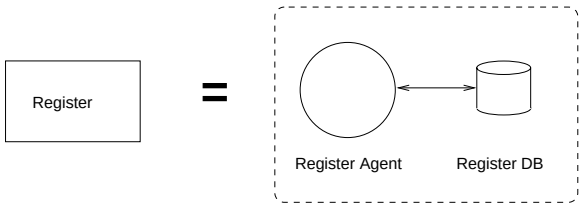


Figure 4.5: Register model

Reduced state. Only certain observable location changes trigger updates. If more fine-grained data is required, additional measurements have to be taken on-demand. This approach reduces the update volume and is therefore suitable if full updates would overload the system while full on-demand measurements are impractical. The GSM system [48] uses the reduced state approach.

The amount of state kept by the location service has a profound effect on its scope and scalability. While many systems assume a characteristic query-to-update ratio, it is possible to dynamically adapt the amount of state according to current load conditions. For example, *caching* is a common technique of optimising access to frequently used data by dynamically creating additional transient state.

4.4.2 State distribution

In our system model (Figure 4.3), we distinguish location service, application agent (LSAA), and sensor agent (LSSA), leaving open the design choice of how state is distributed between location service and external agents. This we refer to as *external state distribution*. Further, the location service itself may be a distributed system. Hence, the *internal state distribution* needs to be considered.

External state can be situated in one of three places:

Sensor-side. Location information can be stored at the point of measurement, such as the GPS receiver. This is practical for positioning systems where located-objects have a dedicated sensor (e.g. GPS), or if only a location monitoring function is required (e.g. for traffic monitoring).

Client-side. Location information can be stored at the point of query, i.e. in the client-side agent. This may compensate for lack of state at the service or sensor level, or may facilitate disconnected operation. However, client-side state can be wasteful if the same location information is stored (and updated) at many clients.

Centralised. Location information can be stored by the shared service facility. Centralised state is often necessary in order to avoid inefficient global broadcasts of queries and updates. Centralised state is typically provided by a logically centralised object register (OR), or a logically centralised location register (LR) (Figure 4.4). In GSM, the OR is implemented centrally in the home location register (HLR). There is no logically centralised LR, although a distributed location register is provided by the VLR.

The character and distribution of logically centralised state greatly influence scalability and performance of a location service. We think of the distribution of internal state as having the orthogonal dimensions of state *replication* and state *partitioning*.

Replication

Replication of state means that multiple copies of state data are maintained at different nodes in a distributed system. Typically, it is workable only if queries to the replicated

data items far outnumber their updates. If updates become too frequent, either availability or consistency suffers. The tradeoff between availability and consistency is determined by the consistency control mechanism. Strong consistency (such as guaranteed by two-phase locking) reduces data availability if updates become too frequent. Weak (i.e. eventual) consistency (such as provided by DNS' replica timeout mechanism) results in incorrect query results if the data is updated too often.

Thus only slowly-changing data items can be replicated with strong consistency. Even then, there need to be compelling reasons to justify the additional communicational and computational complexity. Typical large scale distributed systems, such as DNS or GSM, only guarantee weak replica consistency. This indicates that scalability is much easier to achieve with weak consistency.

Since high-resolution location data may change too quickly to contemplate replication based on multiple identical copies, *multi-resolution replication* needs to be employed. Considering the data items in question (that is location, located-object pairs), there is an obvious correlation between the physical size of the location and the probability that the data item will be updated (i.e. become valid or invalid). Replication of location data at a reduced level of detail is used, for instance, in the GSM system.[1] Alternatively, it is possible to identify dynamically located-objects that either move slowly or whose location is queried often. Then, location information related to those users can be cached. Such a scheme has been proposed for the PCS context [28].

Weakly consistent replication is sometimes combined with *forwarding pointers* [3]. A forwarding pointer is essentially a temporary and unreliable local modification of a single out-of-date replica. It provides a *hint* towards the correct information.

Partitioning

Partitioning of state subdivides state entities into disjoint pieces and assigns them to different nodes in a distributed system. It allows distribution of load resulting from queries and updates among multiple nodes without the constraints of consistency control. This is especially effective if access is evenly spread over all partitions. As a drawback, sometimes a *partition directory* is required in order to find the partition containing a certain piece of state. As far as location information is concerned, the two main approaches to state partitioning are *by location* and *by located-object*.

Combination

Replication and partitioning will normally be combined in order to build hierarchical server structures. Figures 4.6 and 4.7 show two examples.

Partitioning by location means that each node is assigned a coverage area such that it only stores state related to that area (i.e. the locations of located-objects within it). When combined with replication, this naturally leads to hierarchic server structures (Figure 4.6,

[1] The HLR only knows the location area, whereas the VLR has more detailed information ([48], page 102).

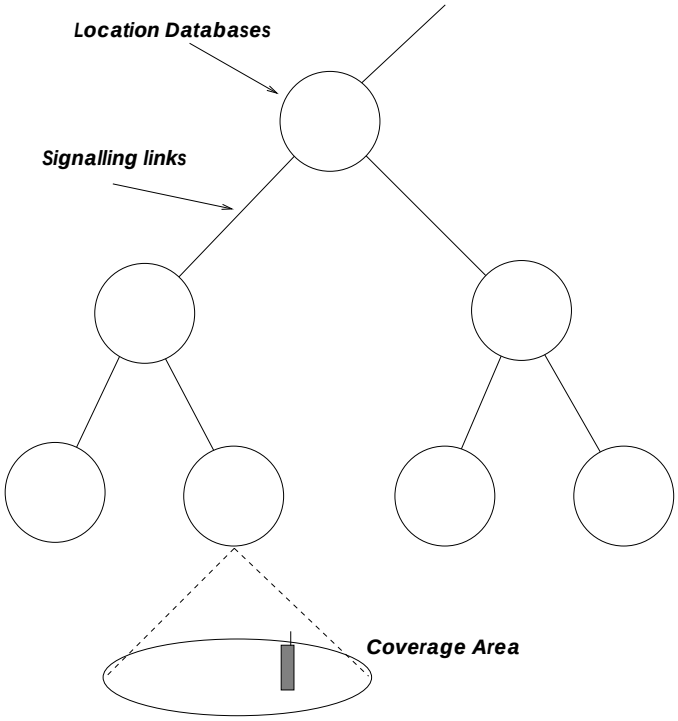


Figure 4.6: State partitioning by location

adapted from [26]). Highly scalable systems, such as mobile telephony systems, have been designed with this pattern. In a two-level form, this approach is used in current mobile communications systems, such as GSM [48]. For better scalability, deeper hierarchies have been proposed, especially for PCS [26, 73].

Partitioning by user (i.e. by located-object) means that one or more located-objects are assigned to a node such that the node only stores data related to those objects. The extreme case is a flat community of user-agents as proposed in [66]. This approach eliminates the need to trust location servers as repositories of private information [68]. Alternatively, a hierarchy of user state nodes can be constructed following some arbitrary structuring of the user space (Figure 4.7). This requires some kind of state replication. Considering the processing updates and queries, this user-partitioned hierarchy is as scalable as the location-partitioned hierarchy described above. However, the latter has the benefit of lower communication costs because location state nodes will be "near" to the location they are responsible for. This cannot be said for user-state nodes.

It should be noted that certain distribution patterns entail additional state ("meta-state"). For example, a partitioned database may need a partition index to avoid exhaustive searches through all partitions. Hence, the amount of state in the system is not independent of distribution pattern.

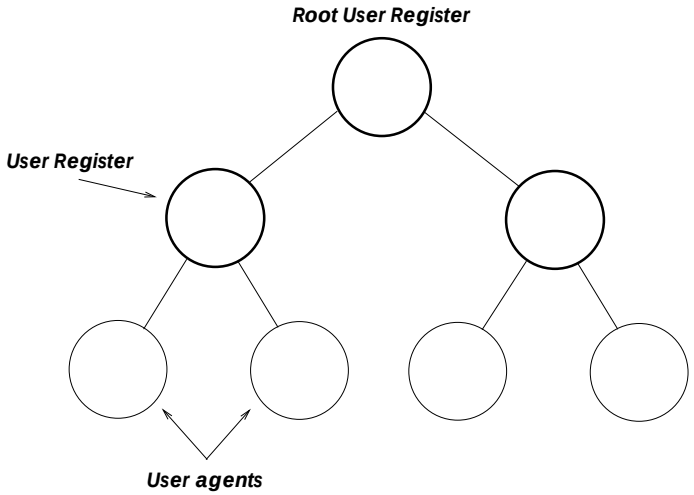


Figure 4.7: State partitioning by located-object

4.4.3 Distribution of processing

Considering Figure 4.3 (page 75), processing of location data can be performed sensor-side, client-side, or somewhere "in the middle". These processing tasks can be subdivided into query-related and update-related processing. While distribution of tasks is closely related to distribution of state, it is possible to migrate processing tasks in exchange for an increased communications cost.

Sensor-side processing involves measuring positions, which can involve complex computations (as in the case of GPS). As more and more processing power becomes available in the form of specialised microprocessors and digital signal processing chips, there can be considerable processing resources available at sensor level. Sensor-level processing is highly desirable because it tends to reduce the bandwidth requirement. However, it is constrained by the amount of information available to a single sensor.

Client-side processing is desirable because it reduces the complexity of the servers' tasks, and thus leads to more light-weight and robust servers. Further, load on the servers is reduced. However, client-side processing may lead to increased message volumes between client and server, because the principal flow of information is from location service to application. Further, the client may be unavailable to receive communications from the server during periods of disconnection.

Centralised processing While sensor-specific computations are best carried out by sensors, and complex client-specific computations by clients (the end-to-end argument put forward in [57]), there often is a set of processing tasks which is so closely coupled to shared state that it cannot be performed efficiently by either sensor or client. Effectively, shared state leads to shared processing. Shared state is often logically centralised, implying logically centralised processing. As with the distribution of location state,

centralised processing facilities can be partitioned according to location or located-object and physically distributed.

4.5 Control flow and synchronisation

A system with a location service may be viewed as a network of processing entities. The mode of communication between these entities ("push"/asynchronous, "pull"/synchronous) is related to state distribution and also affects the functionality available to clients.

Different location sensors support different communication modes: Active Badge sensors with the `abpoll` driver support only "push"-mode, while the UNIX `ruser` RPC service [70] supports only "pull"-mode. On the other hand clients require asynchronous notifications ("push"), queries ("pull"), or both. It appears that reconciling different communication modes is an important challenge of a location service. A location service that builds client "push" on top of sensor "push", or client "pull" on top of sensor "pull" can be implemented easily and without any state. If the communication modes of sensors and clients are mismatched, the service itself needs to hold state. If the client supports only "push", then it is the service's responsibility to obtain updates for centralised state.

4.6 Architectural examples

Having discussed the main design issues for a location service architecture, in this section we examine the architecture of two deployed location services. Then, we propose an architecture for a general, global location service.

4.6.1 Xerox PARC

The location service architecture developed at Xerox PARC [66] follows both partition-by-location and partition-by-user patterns, supporting our belief that a general location service will typically expose both patterns.

As mentioned above, most location state is held by the user agents, which act as partial object registers. They get their information from lower-level location services (Active Badges, `ruser` daemons). These we have labelled as stateless location processors (although they might be PLRs or PORs in their own right). Geographical space is subdivided into regions, each serviced by a location broker, which thus plays the role of a PLR.

No logically centralised OR or LR exist, although their functions could be easily provided by adding an object directory and a location directory to the system (see § 4.6.3 below).

4.6.2 GSM

The GSM architecture [48] basically follows a partition-by-location approach (Figure 4.9). That is, it can be modelled as a tree of nodes each with its own coverage area. Nodes at the same level have disjoint coverage areas, the area of a parent includes the area of

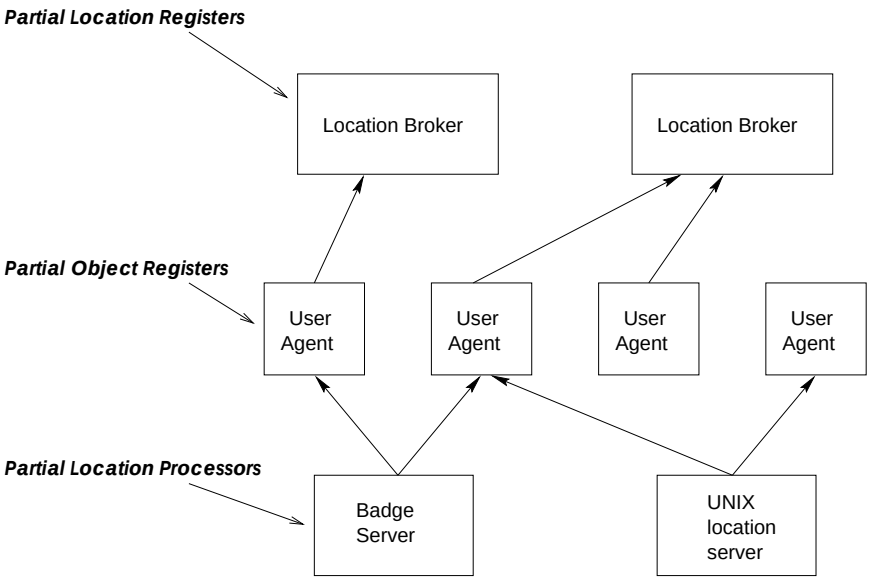


Figure 4.8: Location service architecture of the Xerox PARC ubiquitous computing environment

its children. The root entity is the home location register, which fulfils the function of a central OR. The next level is made up of visitor location registers (VLR, collocated with mobile services switching centres). The VLRs form a distributed LR, with no logical centre (single point of query). Location information is replicated in HLR and VLR, with the VLR having more detailed location information but only a subset of the HLR's located-objects. The replica control scheme effectively supports eventual consistency between HLR and VLRs (see [48], page 471). Other mobile telephone standards, such as IS-41, follow a similar approach for replica control [27]. The nodes further down the tree, base station controllers and transmitters, do not explicitly store location information and are therefore shown as stateless location processors.

The mobile terminal, i.e. GSM phone, monitors available cells and decides which to use. Thus we can say that the phone hosts the location service sensor agent LSSA. The phone itself cannot access the location information stored in HLR or VLRs, that is, it cannot actually host a client to the location service. Location-aware applications, or at least their agents, LSAA, must run on the fixed network (for example, in the mobile switching centre). Such applications include the setup of mobile-terminating calls, and the generation of location-dependent traffic reports.

4.6.3 A global location service

The above sections have discussed general aspects of location service, in conjunction with a discussion of existing architectures. We now propose a novel architecture that overcomes some of the shortcomings of existing approaches. The principal aim is to achieve a scalable design for multiple, non-hierarchic administrative domains. Further, we do not wish to tie the architecture to either user-agent or location-server approach, and both should be

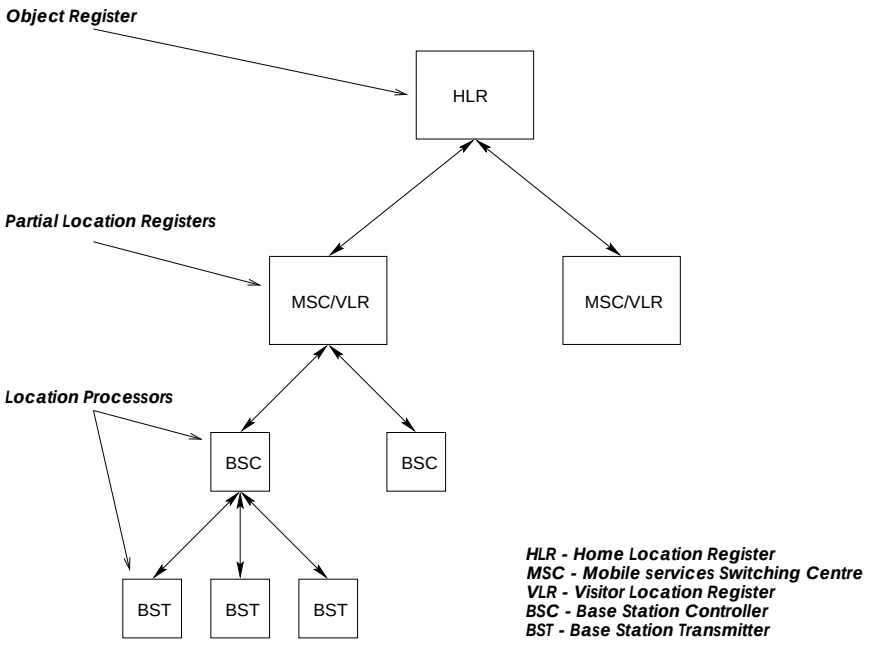


Figure 4.9: Location management architecture of GSM

allowed to co-exist.

The principal idea of the architecture is a flexible partitioning and distribution scheme for both OR and LR. We achieve this by introducing logically centralised directories for located-objects and locations, *object directory* (OD) and *location directory* (LD), respectively (Figure 4.10). These provide a layer of indirection that hides physical location and structure of user-related and location-related information in the system. It does not matter, for example, whether user-related information is stored by a user agent or by a central HLR. As a result, a single framework can cover an architecturally heterogeneous system. The functionality of OD and LD could also accommodate mobile agents for located-objects and locations.

Figure 4.11 gives an overview of our architecture. The dotted lines show administrative boundaries. The service consists of a central directory for both located-objects and locations, plus a number of independent location service providers. The central directory must know the service providers for all located-objects and locations in the system. This implies that if a located-object's (or a location's) agent permanently moves to another service provider, the central directory has to be updated. We also anticipate that the location directory is fully aware of the hierarchical structure of the location space such that queries over multiple location service providers can be correlated.

Each location service provider covers a subset of sensors, located-objects, and locations. In the extreme case, a location service provider could cover a single location or located-object. Located-objects, locations and sensors must be allowed to overlap between service providers. Located-objects may be allowed to roam between location service providers. In this case, a hand-off procedure between service providers would have to be executed. In order to avoid frequent updates of the central directory, stationary or slowly-moving *home*

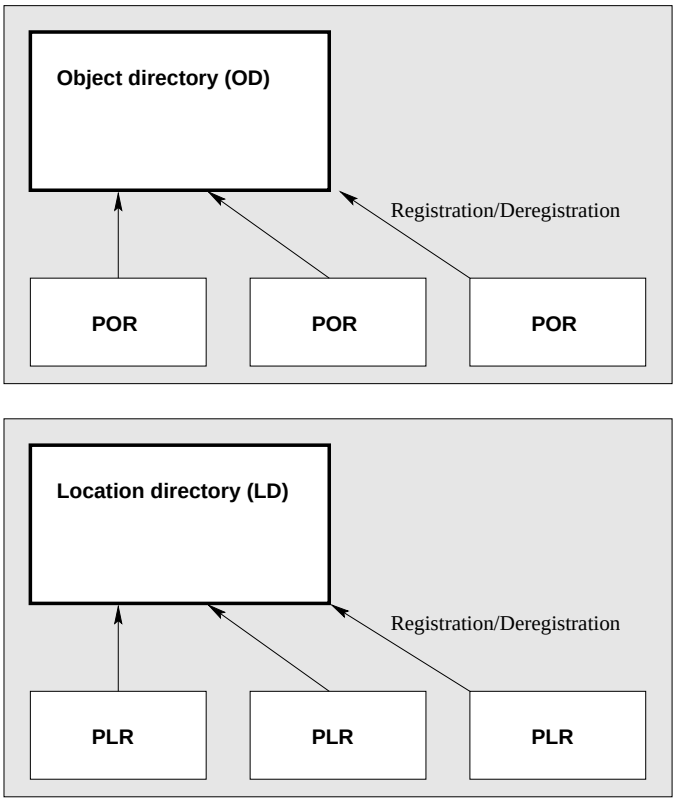


Figure 4.10: Distributed location and object registers

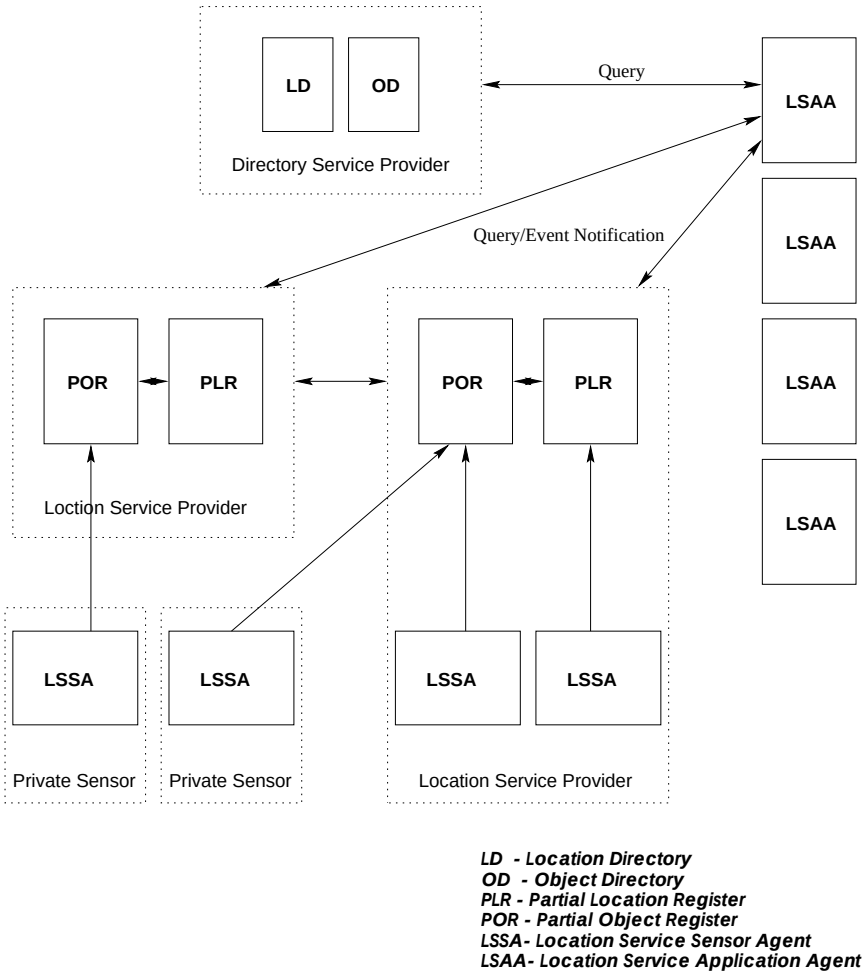


Figure 4.11: Proposed architecture for a global general-purpose location service

agents would be employed.

The sensor agents (LSSA) are each connected to either a PLR or a POR of a location service provider, depending on whether the sensor is location-centric (e.g. an Active Badge sensor), or user-centric (e.g. a GPS receiver). Sensor and sensor agent do not necessarily belong to the location service provider.

In this architecture, the location service application agent (LSAA) has the task of integrating data gathered from different components. Firstly, the LSAA needs to consult the directory before performing any query regarding a location or located-object. Secondly, the LSAA must recognise and reconcile overlaps between service providers. For example, in order to count the number of people in a certain building, it may be necessary to consult multiple service providers and eliminate any duplicated results.

We have conducted some experiments with this approach in the context of an "Active Office" location service (cf. § 5.8). There, a client program connects to two overlapping location services, and integrates the results. This is feasible as long as there is a directory relating the respective location models. Fusing heterogeneous asynchronous event notifications appears to be the most significant difficulty.

We envisage that propagation of updates and events between service providers and application agents would be supported by an underlying hierarchical event channel. This infrastructure adaptively instantiates and garbage-collects sub-channels for events related to particular sets of located-objects and locations. Equally, such a support platform could selectively employ lower-level broadcast protocols when they are more efficient than point-to-point communication.

Our approach provides a framework for integrating heterogeneous location services globally. The architecture separates concerns: global scope is provided by the central directories (LD/OD), functionality is supported by the location service providers, and communicational complexity is addressed by the underlying event distribution infrastructure. Therefore, we think that this is a promising approach to provide a global, scalable, and general-purpose location service.

4.7 Chapter Summary

In this chapter we have defined the notion of location service: a shared object providing information about the physical location of located-objects. We have analysed which resources would be encapsulated and managed by such a service, and which additional service logic is required. Having thus clarified the scope of functionality of such a service, we proceeded to identify the basic dimensions of its functionality. On this basis, we proposed a functional decomposition (Figure 4.1 page 72). The basic service functionality is supported by a formal specification and a corresponding implementation (appendix D).

Since the functionality of a service constitutes only a part of its design, we have proposed a system model addressing both structural layering and structural distribution within the service, closely following the system modelling approach of the X.500 directory service. Based on our model, we have identified the basic dimensions of the design of a distributed architecture: amount and distribution of state, distribution of processing, and

control flow. By applying these dimensions to existing location services, we have derived the fundamental patterns of partitioning by location and user.

Our architectural considerations were concluded by three examples, one of which is our proposal for a general and global location service. Our approach is based on a federation of heterogeneous location service providers, with centralised support functions such as directories and an event delivery service. By accommodating heterogeneous location service subsystems we recognise that no single architecture or design is appropriate for all located-objects or for all locations. Initial validation for this approach is provided by the "Active Office" location service described in § 5.8.

Chapter 5

Acquisition of Location Data

The acquisition function of a location service provides a layer of abstraction that allows higher functional layers to be implemented independently of concrete location sensor technologies. In this chapter, we discuss the requirements for the structural and functional design of the acquisition function. We propose an abstract three-layer stack, which can be mapped to a number of architectures to provide the acquisition function. Further, we discuss two concrete algorithms for acquiring symbolic location sightings. Finally, we apply our functional abstractions to existing location service implementations.

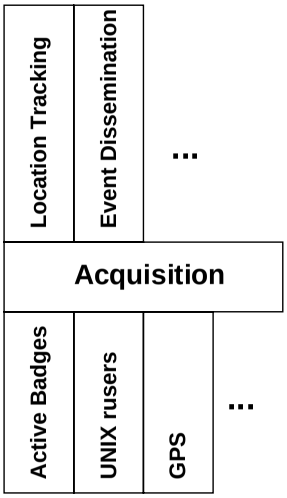


Figure 5.1: Architectural context of the acquisition layer

Figure 5.1 shows the position of the acquisition function in the location service architecture. Accepting input from a variety of location sensor systems, the acquisition layer provides a sensor-independent platform for higher-level processing and dissemination functions.

5.1 Requirements

Considering the global requirements for the acquisition function identified in § 2.1 (page 21), spatio-temporal resolution, real-time delivery, global coverage, and openness/generality are particularly relevant.

Spatio-temporal resolution This is a requirement that needs to be addressed by the functional design. It appears that location-aware applications often require more detailed information than is supported by location-sensor technology. Conversely, if information is available, there is bound to be an application that uses it. Hence, the acquisition layer should provide the means to achieve the maximum spatio-temporal resolution that is supported by the input from the location sensors. Nevertheless, it is desirable to adapt acquisition accuracy to application requirements in order to avoid unnecessary overheads.

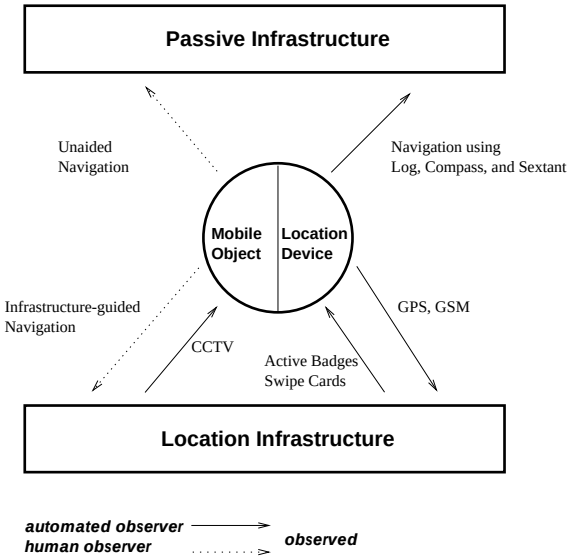


Figure 5.2: Patterns of tracking and positioning

Openness and generality Figure 5.2 demonstrates the heterogeneity of the problem domain, by showing that location sensors can be part of the infrastructure and/or the mobile object. Further, there are various low-level control and data flows. The generality requirement means covering important approaches to location sensing, while openness means that new sensing technologies can be integrated as they are developed. This implies a general functional design and an open architecture. In this chapter, we shall concentrate on functional generality.

Real-time delivery of information If the location service as a whole needs to support real-time information delivery, this must be based on real-time low-latency acquisition of information, and needs to be supported by the structure (architecture) of the acquisition layer. In this chapter, we shall concentrate on functional issues enabling real-time

information delivery if an appropriate architecture is chosen.

Global coverage This requirement overlaps with the generality/openness requirement in that global coverage requires dealing with heterogeneous sensors systems. Additionally, there is an implied requirement for a scalable architecture that can handle a large number of sensors and an even larger number of located-objects. This is an architectural issue that will be discussed in the following chapters.

5.2 Design considerations

Given the requirements elaborated above, there are a number of functional and architectural dimensions to be considered when designing the acquisition layer.

5.2.1 Functional design issues

The internal data model is perhaps the most important design choice from the point of openness and generality. To solve the problem of **n** heterogeneous inputs and **m** heterogeneous outputs the following solutions appear applicable:

- Choose an abstract internal representation. Then, $\mathbf{m} + \mathbf{n}$ translation functions are needed.
- Use external formats internally. Provide $\mathbf{m} \cdot \mathbf{n}$ translation functions to convert data.

As far as openness and generality are concerned, the first solution is preferable for extensibility and independence from sensor-technology. Elements of the second approach may be applicable for special-interest location data, that is, for data provided only by a few sensors and used by a few applications.

Should the internal data model be based on geometric or symbolic locations? In a heterogeneous environment, it seems that a hybrid location model is appropriate. Perhaps surprisingly, there is also an architectural dimension to this choice. Geometric models seem to be a good match for continuous stateful positioning systems, whereas symbolic models are well suited for stateless event-driven systems.

Another interesting choice is how to treat spatially and temporally overlapping information. Such overlaps are likely if heterogeneous sensor technologies are chosen. While overlaps make the processing of location data more complicated, they also present the opportunity to detect inconsistencies and improve accuracy. Hence, overlaps can enhance correctness and completeness of location data.

5.2.2 Architectural design issues

Although we are not going to propose a particular architecture for the acquisition function in this chapter, it is valuable to look at some of the issues involved to provide a background for the functional design.

- *Positioning* vs. *Tracking*. An acquisition function can be used in order to build either a positioning or a tracking system. A positioning system measures its *own* location with the help of the infrastructure (e.g. vehicle navigation systems [85]). A tracking system measures the location of *other* located objects (e.g. the Active Badge System [75]). Different architectures are required for each case, although the processing functions may be very similar.

- *Local* vs. *Remote measurement*. Tracking systems may be built on top of positioning systems, and vice versa. In such cases, the acquisition layer has the task of providing a specialised location transparency. This also requires physical and logical distribution of the acquisition function.

- *Synchronous* vs. *Asynchronous dissemination*. Applications are likely to require both events and polling, while location sensors may only support one of the two.

- *Discrete update* vs. *Continuous change*. Location sensing is mostly opportunistic (i.e. discrete), with the exception of integrated multi-sensor positioning systems which can provide continuous information (see [85]).

- *Stateful* vs. *Stateless sources*. Location sources can also be classified into *stateless* and *stateful* sources. Stateless sources need to wait for the underlying hardware to supply data, whereas stateful sources can provide information (i.e. current state) continuously.

- *Homogeneous* vs. *Heterogeneous sensors*. Sensor types are often complementary in their spatial and temporal coverage. Also, dissimilarity of their error profiles makes simultaneous failure unlikely. Therefore the architecture of the acquisition function must be designed to cope with a variety of sensor types with different computational and communication characteristics.

Considering these dimensions of the architectural design space, we have identified two points (i.e. designs) that appear to be "natural" architectures for *acquisition* systems:

1. The stateless infrastructural location tracking system. Such a system uses a symbolic location model to provide discrete updates via asynchronous events. This system would be distributed covering many sensors of the same type. For example, the Active Badges use this approach.

2. The stateful positioning system. This system would use an integrated multi-sensor system to provide continuously changing geometric location measurements, which can be retrieved at any point in time through synchronous queries. The system would be mostly local using a mix of sensor technologies. Such an approach is used today for in-car navigation systems.

This idea of a "match" between location model, information dissemination, state, and distribution is certainly supported by our experience with building location services. Things tend to become complicated if both approaches need to be combined. However,

combination is necessary in order to build an acquisition function for a general-purpose location service.

In our approach, we aim to provide the end-user functionality of the second approach while using elements of the first approach internally. For the acquisition function, this means focusing on event-based stateless information processing.

5.3 Functional structure

The acquisition component has to collect data from all the sensors in the system and present it in a unified way to higher levels of the application.

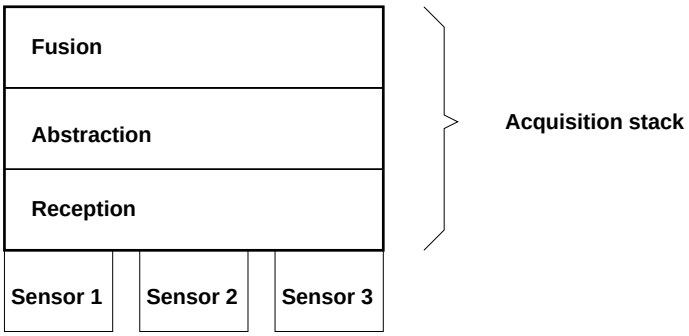


Figure 5.3: Simple acquisition stack

We model the acquisition module as stack of layers (Figure 5.3). There are three principal layers: reception, abstraction, and fusion.

These three layers form the *acquisition stack*. As shown in (Figure 5.4), a single acquisition stack can receive input from either sensors or from other acquisition stacks. Hence, the architecture is recursive, allowing for multi-stage acquisition trees. This structure should also aid the partitioning and distribution of the acquisition function. Similarly, openness is facilitated by this non-monolithic approach.

The acquisition stack model is architecture-independent and can be expressed in a variety of architectural styles [62], such as pipe/filter, layered, event-based, or object oriented system. Below is a brief description of each of the layers as shown in Figure 5.3.

Sensors

The sensor layer comprises hardware and low-level software (firmware) that is responsible for operation of the tracking or positioning sensors. This is where low-level communication protocols, such as the badge–sensor protocol of the Active Badge system or the satellite–to–receiver protocol of the GPS system would be implemented.

Since such systems tend to be proprietary or vendor-specific, we cannot make any assumption except that there is a communication protocol or API allowing the sensor layer to be connected to the location service.

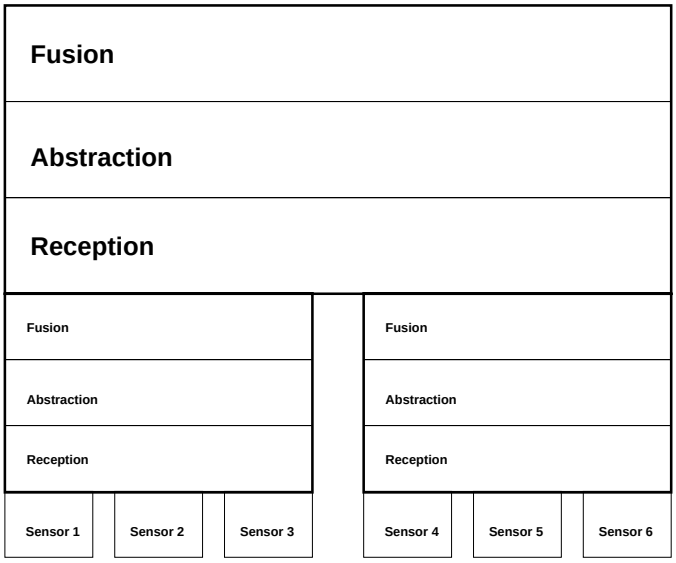


Figure 5.4: Tree of acquisition stacks

Reception layer

The reception layer supports distribution and synchronisation transparency by providing a "sensor bus", a substrate for communication with location sensors.

Sensors may be attached to a set of sensor-dependent locations (e.g. radio cells), or to a located-object. The acquisition layer removes dependencies on the identity of the sensor. If a sensor is a positioning sensor rather than a tracking sensor, the reception layer hides this by employing some wireless communications medium.

Abstraction layer

The abstraction layer unifies the sensors' data representations. It therefore needs sensor-independent data and confidence models. Even if there is only one sensor type in the system, e.g. badge sensors, it is useful to have an abstraction layer which hides, for example binary identifiers or low-level confidence metrics. Such an approach keeps the system open to addition of new location tracking systems as they become available.

Fusion layer

The fusion layer correlates the sightings belonging to the same located-object from various sources. In this chapter, we only discuss the fusion of sightings at a single point in time.[1] Since the data provided by the abstraction layer already has a homogeneous representation, the main issues of fusion are to exploit overlap and to detect inconsistencies.

[1] Fusion over the temporal domain is discussed in chapter 6 (page 111).

5.3.1 Layer mapping

The functional layers of the acquisition stack can reside on different processing entities in the system: on the sensor, within the service, or in the client. These mappings are not mutually exclusive.

- Sensor-level acquisition functions can create multi-sensor systems that integrate primitive sensors of different types. To the outside world, the composite sensor looks like a normal sensor with improved accuracy, coverage, and fault-tolerance.
- Service-level acquisition is performed within the location service to combine input from multiple sensors relating to a single located-object. The aims are to shield the application from the sensor details and to reap the benefits of multi-sensor integration.
- Client-level abstraction and fusion is carried out at the application level. This is undesirable in most cases because it implies exposure to the heterogeneity of location sensors. However, sometimes the client has additional, application-specific location data that need to be combined with the input from the location service.

In the following sections, we present a mathematical model detailing data flow between, and processing within, the layers of the acquisition stack.

5.4 Reception

The reception layer collects location events from various sensors. Therefore, its main functional task is to resolve dependencies of single sensor contexts. For example, a sensor may report on a number of its sub-locations. Knowledge of a sub-location is only useful when the sensor itself is also known.

To characterise the functionality of each layer, we use a simple calculus of mathematical functions. We use the following notational conventions: τ denotes a sensor type, σ denotes a sensor. S denotes a sensor identifier, L a location identifier, T identifies a certain time (point or interval), O is a object identifier. Superscripts denote context dependencies, e.g. O^{τ} means object identifier O as seen by sensors of type τ. Subscripts denote indices.

Input We model the primary input of the acquisition stack as a set of events E. This does not restrict the interaction style, since each location event could be delivered using a variety of methods, including request-reply (pull) and groupcast (push).

The data exchanged between sensors and the reception layer has the general form:

$$\begin{aligned} \mathrm{E} &= \{\mathrm{E}_1, ..., \mathrm{E}_\mathrm{n}\} \\ &= \{(\sigma_1, \tau_1, \mathrm{T}_1^{\sigma}, \mathrm{L}_1^{\sigma}, \mathrm{O}_1^{\sigma}), ..., (\sigma_\mathrm{n}, \tau_\mathrm{n}, \mathrm{T}_\mathrm{n}^{\sigma}, \mathrm{L}_\mathrm{n}^{\sigma}, \mathrm{O}_\mathrm{n}^{\sigma})\} \end{aligned}$$

$\sigma^\tau_{\mathbf{i}}$ is the sensor identifier for a sensor of type τ, $\mathrm{T}^\tau_{\mathbf{i}}$ is the sensor-type-dependent timestamp (which may be missing), $\mathrm{L}^\sigma_{\mathbf{i}}$ is the location datum, $\mathrm{O}^\sigma_{\mathbf{i}}$ is the identifier of the located-object. Often only either $\mathrm{L}^\sigma_{\mathbf{i}}$ or $\mathrm{O}^\sigma_{\mathbf{i}}$ will be provided since they are relative to the sensor. Both data items can contain auxiliary data, such as confidence, velocity, or direction. All the data exchanged depends on the sensor type σ.

Processing and Output The processing functionality is characterised by the reception function **recept**, which is applied to each incoming event. This function translates sensor-dependent identifiers of objects and locations into sensor-independent identifiers (which are still relative to sensor type τ):

$$\begin{aligned}
\mathbf{recept}: &\quad (\sigma, \tau, \mathrm{T}^\sigma, \mathrm{L}^\sigma, \mathrm{O}^\sigma) \mapsto (\tau, \mathbf{time}^\sigma(\mathrm{T}^\sigma), \mathbf{loc}^\sigma(\mathrm{L}^\sigma), \mathbf{obj}^\sigma(\mathrm{O}^\sigma)) \\
\mathbf{loc}^\sigma: &\quad \mathrm{L}^\sigma \mapsto \mathrm{L}^\tau \\
\mathbf{obj}^\sigma: &\quad \mathrm{O}^\sigma \mapsto \mathrm{O}^\tau \\
\mathbf{time}^\sigma: &\quad \mathrm{T}^\sigma \mapsto \mathrm{T}^\tau
\end{aligned}$$

Also the timestamp can contain dependencies of the sensor instance. For example, sensors might be situated in different time zones but report sightings stamped with the local time. Since the dependency on the sensor instance has been removed, $\sigma^\tau_{\mathbf{i}}$ is now redundant.

The acquisition layer applies the acquisition function to each event in turn:

$$\begin{aligned}
\mathrm{E}' &= \{\mathrm{E}'_1, ..., \mathrm{E}'_\mathrm{n}\} \\
&= \{\mathbf{recept}(\mathrm{E}_1), ..., \mathbf{recept}(\mathrm{E}_\mathrm{n})\} \\
&= \{\mathbf{recept}((\sigma_1, \tau_1, \mathrm{T}^\sigma_1, \mathrm{L}^\sigma_1, \mathrm{O}^\sigma_1)), ..., \mathbf{recept}((\sigma_\mathrm{n}, \tau_\mathrm{n}, \mathrm{T}^\sigma_\mathrm{n}, \mathrm{L}^\sigma_\mathrm{n}, \mathrm{O}^\sigma_\mathrm{n}))\} \\
&= \{(\tau_1, \mathrm{T}^\tau_1, \mathrm{L}^\tau_1, \mathrm{O}^\tau_1), ..., (\tau_\mathrm{n}, \mathrm{T}^\tau_\mathrm{n}, \mathrm{L}^\tau_\mathrm{n}, \mathrm{O}^\tau_\mathrm{n})\}
\end{aligned}$$

As a result, the reception layer produces a set of events independent of the sensor instances where they originated.

5.5 Abstraction

The task of the abstraction layer is to map all sightings into a single, unified, representation domain. This applies mainly to representations of identifiers for located-objects and symbolic locations, but also the representation of time values may be sensor-type-dependent, in which case time values need to be translated, too. The abstraction function is applied to each event in turn.

Input As produced by the reception layer, the abstraction layer accepts a set E' of sensor-independent events $\mathrm{E}'_\mathbf{j}$. Each of the events must be interpreted against the context of the type of sensor that produced it.

Processing and Output Similarly to the reception function, there is an abstraction function which translates sensor-type-dependent identifiers into an abstract representation. The translation function is selected by sensor type τ.

$$\begin{aligned}
\textbf{abstr}: &\quad (\tau, \mathrm{T}^\tau, \mathrm{L}^\tau, \mathrm{O}^\tau) \mapsto (\textbf{time}^\tau(\mathrm{T}^\tau), \textbf{loc}^\tau(\mathrm{L}^\tau), \textbf{obj}^\tau(\mathrm{O}^\tau)) \\
\textbf{time}^\tau: &\quad \mathrm{T}^\tau \mapsto \mathrm{T} \\
\textbf{loc}^\tau: &\quad \mathrm{L}^\tau \mapsto \mathrm{L} \\
\textbf{obj}^\tau: &\quad \mathrm{O}^\tau \mapsto \mathrm{O}
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}'' &= \{\mathrm{E}''_1, ..., \mathrm{E}''_\mathrm{n}\} \\
&= \{\textbf{abstr}(\mathrm{E}'_1), ..., \textbf{abstr}(\mathrm{E}'_\mathrm{n})\} \\
&= \{\textbf{abstr}((\tau_1, \mathrm{T}^\tau_1, \mathrm{L}^\tau_1, \mathrm{O}^\tau_1)), ..., \textbf{abstr}((\tau_\mathrm{n}, \mathrm{T}^\tau_\mathrm{n}, \mathrm{L}^\tau_\mathrm{n}, \mathrm{O}^\tau_\mathrm{n}))\} \\
&= \{(\mathrm{T}_1, \mathrm{L}_1, \mathrm{O}_1), ..., (\mathrm{T}_\mathrm{n}, \mathrm{L}_\mathrm{n}, \mathrm{O}_\mathrm{n})\}
\end{aligned}$$

Abstraction is applied to individual events. Hence, the output of the abstraction layer is a set of abstracted events. The representation of the output events is free from sensor dependencies.

5.5.1 Translation properties

At this point, we would like to devote some attention to the translation functions (**time**, **loc**, **obj**). To be useful, these functions satisfy correctness and completeness properties.

Let $\mathbf{f}^\tau \in \{\textbf{time}^\tau, \textbf{loc}^\tau, \textbf{obj}^\tau\}$ be a translation function, and representation domains D^τ and $\mathrm{D}_{\textbf{abstr}}$, such that $\mathbf{f}^\tau\colon \mathrm{D}^\tau \to \mathrm{D}_{\textbf{abstr}}$. We compose all $\mathbf{f}^\tau$ for a set of sensor types T into a single function:

$$\mathbf{f}_{\textbf{abstr}}\colon (\tau, \mathrm{d}^\tau) \mapsto \mathbf{f}^\tau(\mathrm{d}^\tau),\ \mathrm{d}^\tau \in \mathrm{D}^\tau \text{ and } \tau \in \mathrm{T}$$

Intuitively, a correct translation does not change the semantic meaning of the translated symbols. This notion of correctness is mathematically expressed as a *homomorphism*, a relationship-preserving mapping. In our case, the relevant relationships are *semantic equivalence* $=_\mathbf{s}$ (relating symbols corresponding to the same real-life entity), and *semantic ordering* $\leq_\mathbf{s}$ (relating symbols which refer to objects that are semantically ordered).

Property 1 *A translation function* $\mathbf{f}_{\textbf{abstr}}$ *is* correct, *if, and only if, it satisfies the following constraints:*

$$(\forall\, \tau_1, \tau_2 \in \mathrm{T}).(\forall\, \mathrm{d}^{\tau_1} \in \mathrm{D}^{\tau_1}).(\forall\, \mathrm{d}^{\tau_2} \in \mathrm{D}^{\tau_2}).(\mathbf{f}_{\textbf{abstr}}(\tau_1, \mathrm{d}^{\tau_1}) =_\mathbf{s} \mathbf{f}_{\textbf{abstr}}(\tau_2, \mathrm{d}^{\tau_2})) \Leftrightarrow (\mathrm{d}^{\tau_1} =_\mathbf{s} \mathrm{d}^{\tau_2})$$

$$(\forall\, \tau_1, \tau_2 \in \mathrm{T}).(\forall\, \mathrm{d}^{\tau_1} \in \mathrm{D}^{\tau_1}).(\forall\, \mathbf{d}^{\tau_2} \in \mathrm{D}^{\tau_2}).(\mathbf{f}_{\textbf{abstr}}(\tau_1, \mathrm{d}^{\tau_1}) \leq_\mathbf{s} \mathbf{f}_{\textbf{abstr}}(\tau_2, \mathrm{d}^{\tau_2})) \Leftrightarrow (\mathrm{d}^{\tau_1} \leq_\mathbf{s} \mathrm{d}^{\tau_2})$$

□

Further, translation functions may be required to provide a *lossless* mapping. Loss of information happens if two semantically distinct values are mapped to the same value in the abstract domain. For example, a mapping translating both `Ale` and `Lager` into `Beer` is semantically lossy.

Property 2 *A translation function* $\mathbf{f_{abstr}}$ *is* complete*, if, and only if, it satisfies the following constraint:*

$$(\forall\, \tau_1, \tau_2 \in \mathrm{T}).(\forall\, \mathrm{d}^{\tau_1} \in \mathrm{D}^{\tau_1}).(\forall\, \mathrm{d}^{\tau_2} \in \mathrm{D}^{\tau_2}).(\mathbf{f_{abstr}}(\tau_1, \mathrm{d}^{\tau_1}) = \mathbf{f_{abstr}}(\tau_2, \mathrm{d}^{\tau_2})) \Rightarrow (\mathrm{d}^{\tau_1} =_{\mathbf{s}} \mathrm{d}^{\tau_2})$$

□

Note that if $\mathbf{f_{abstr}}$ is *bijective* (one-to-one), completeness follows trivially. However, often a bijective mapping will be impossible because different symbols can refer to the same real-life object. It is even desirable to have *at most one* abstract symbol for each real-life entity. This non-duplication property is the converse of the completeness property given above.

Property 3 *A translation function* $\mathbf{f_{abstr}}$ *is* non-duplicating*, if, and only if, it satisfies the following constraint:*

$$(\forall\, \tau_1, \tau_2 \in \mathrm{T}).(\forall\, \mathrm{d}^{\tau_1} \in \mathrm{D}^{\tau_1}).(\forall\, \mathrm{d}^{\tau_2} \in \mathrm{D}^{\tau_2}).(\mathrm{d}^{\tau_1} =_{\mathbf{s}} \mathrm{d}^{\tau_2}) \Rightarrow (\mathbf{f_{abstr}}(\tau_1, \mathrm{d}^{\tau_1}) = \mathbf{f_{abstr}}(\tau_2, \mathrm{d}^{\tau_2}))$$

□

These are the formal properties of translation functions. For the design of the acquisition function, correctness is crucial. Non-duplication is also important. It should be noted that, in practice, semantic mappings (especially of identifiers for located-objects) may be dynamic. This makes correctness and non-duplication more difficult to achieve. On the other hand, lossy translation may be acceptable in some cases, especially if the raw data has an unnecessary degree of accuracy.

To design a translation function, one would typically start by specifying the target domain $\mathrm{D}_{\mathbf{abstr}}$. The following options may be considered:

- Choose a sensor-type dependent representation domain D^{τ_i} as the target representation domain $\mathrm{D}_{\mathbf{abstr}}$. Thus, the abstraction function $\mathbf{f_{abstr}}$ translates all inputs from other domains values into the representation domain D^{τ_i}. This approach can be chosen if there is one primary input which is only augmented by other sources. However, the openness and flexibility of the system are restricted.

- Design a target representation domain $\mathrm{D}_{\mathbf{abstr}}$ independent of the sensor-types being used. The construction of $\mathrm{D}_{\mathbf{abstr}}$ should be guided by application requirements. Then mappings for all the sensor-dependent input domains need to be defined. This approach implies more effort, but results in a more general and more open system.

In both cases, the choice of $\mathrm{D}_{\mathbf{abstr}}$ determines the granularity of information that will be available to applications *and* the end-user. We can either strive to preserve all information

from all inputs, or preserve only information required by applications. Since both criteria are likely to change when the system is deployed, $D_{\mathbf{abstr}}$ and $\mathbf{f_{abstr}}$ should be easily reconfigurable.

5.6 Fusion

In this context, fusion is concerned with merging inputs related to a *single object* for a *single point in time*. Fusion across the temporal domain is discussed in chapter 6.

Input The input is a set E'' of abstract sighting events as provided by the abstraction layer:

$$\begin{aligned} E'' &= \{E''_1, ..., E''_n\} \\ &= \{(T_1, L_1, O_1), ..., (T_n, L_n, O_n)\} \end{aligned}$$

Processing and Output The fusion layer must integrate related sightings events for a point in time, resulting in a set of confidence-weighted location values. Processing consists of the following tasks:

1. Identify the points in time for which to perform the fusion. This can be driven by application requests or by sighting events received from location sensors.
2. Group the input events from the input stream E'' according to their relation to individual located objects. This may require knowledge of dynamic relationships between located-objects (for example, between a badge and its wearer).
3. Integrate previously identified groups of related sightings. Conceptually, this process compounds the information from each group into a single piece of data. Due to incomplete and inconsistent information, this piece of data is likely to be a *confidence function*.

The first two tasks are modelled by the relevance function **relevant**, aided by the function **obj** to map object aliases. The third task is modelled by the function **fusion**:

$$\begin{aligned} \textbf{fusion}: &\quad \{(L_1, O_1), .., (L_n, O_n)\} \mapsto (L \mapsto c) \\ \textbf{relevant}: &\quad (T, O, E'') \mapsto \{(L, O_\mathbf{i}) | (T, L, O_\mathbf{i}) \in E'' \wedge O_\mathbf{i} \in \mathbf{obj}(O)\} \\ \textbf{obj}: &\quad O \mapsto \{O_1, ..., O_n\} \end{aligned}$$

The **fusion** function produces a confidence function that weights each location value L with a confidence value c. The confidence function describes the amount of incorrectness and incompleteness detected by the fusion function.

Confidence values must be partially ordered. This allows for multi-dimensional confidence metrics that would be excluded by a requirement for total ordering:

Property 4 *The set* C *of all possible confidence values* **c** *is an* **irreflexive partial order.**

$$(\forall\, c_1, c_2, c_3 \in C).(c_1 < c_2 \wedge c_2 < c_3 \Rightarrow c_1 < c_3)$$

$$(\forall\, c_1, c_2 \in C).(c_1 < c_2 \Rightarrow \neg\, c_2 < c_1)$$

□

Further, each confidence function **f** resulting from a fusion must be *monotonic* with respect to the partial ordering of locations:

Property 5 *For all confidence functions* $\mathbf{f}: L \mapsto c$ *the following property holds:*

$$(\forall\, L_1, L_2).(L_1 < L_2 \Rightarrow \mathbf{f}(L_1) \leq \mathbf{f}(L_2))$$

This property ensures that the confidence weighting has a degree of semantic consistency. That is, confidence that an object is in a certain location cannot decrease if we expand that location.

Armed with the function described above, the fusion stage can now be described as a function mapping the input event set E'' into the output event set E''':

$$\begin{aligned} E''' &= \{E'''(O_\mathbf{i}) | O_\mathbf{i} \in \mathbf{dom\ obj}\} \\ E'''(O) &= \{E'''(T_\mathbf{i}, O) | (\exists\, \mathbf{e}).(\mathbf{e} \in \mathbf{relevant}(T_\mathbf{i}, O, E''))\} \\ E'''(T, O) &= \mathbf{fusion}(\mathbf{relevant}(T, O, E'')) \end{aligned}$$

As a result of fusion, there is one confidence function per object for each point in time where a related location event has occurred.

5.7 Acquisition algorithms

The previous sections have presented a high-level view of the structure of the acquisition stack and the functions performed in each layer. In this section, we describe two concrete acquisition algorithms for discrete symbolic location data. The first algorithm was proposed by Rizzo *et al.* [55]. This we use as a background to propose our own acquisition algorithm, an earlier version of which has been propounded in [34].

5.7.1 Attribute matching

In the context of the Active Office project described in § 2.5.3 (page 32), an attribute-based approach to abstraction and fusion of location data has been proposed. We shall use our notation to describe this algorithm.

Location model Locations are modelled by a set of attribute-value assertions. For example, a location associated with a room would have attributes *room number*, *telephone extension*, *workstation name*, and *badge sensor*. Thus, the identity of location tracking

sensors is specified by attributes of the locations associated with their coverage area. This location model has the advantage that additional location-dependent information (such as room number, telephone extension) can be readily accommodated. Further, multi-valued attributes (i.e. attributes with lists of values) can be used to model overlapping locations.

Let L be the set of all defined locations in the system. L is stored by a *location directory*. Let A^* be the set of all attributes that are used to describe locations $l \in L$. Further, let L^* be the set of all locations that can be described using A^*. Since also L is defined in terms of attributes from A^*, L must be a subset of L^*.

If $\mathbf{A}(\mathbf{l})$ denotes the set of attributes of a location $\mathbf{l}$, and $\mathbf{a}(\mathbf{l})$ the set of values of attribute $\mathbf{a}$, Rizzo *et al.* have defined *equivalence* $\sim$ between locations as follows ([55], page 5):

$$\begin{aligned} l_1 \sim l_2 &\Leftarrow \neg\,(A(l_1) \cap A(l_2) = \emptyset) \wedge (\forall\, a \in A(l_1) \cap A(l_2)).(\neg\,(a(l_1) \cap a(l_2) = \emptyset)) \\ \neg\, l_1 \sim l_2 &\Leftarrow \neg\,(A(l_1) \cap A(l_2) = \emptyset) \wedge (\exists\, a \in A(l_1) \cap A(l_2)).(a(l_1) \cap a(l_2) = \emptyset) \end{aligned}$$

The definition leaves us with pairs of locations whose equivalence we cannot establish. Further, the defined relationship is not an equivalence relationship in the mathematical sense because it is not transitive. However, transitivity can be established if no overlaps between locations are allowed.

A key notion in the algorithm is *expansion* of locations with the help of the location directory:

$$\mathbf{expand} : L^* \rightarrow L$$

$$(\mathbf{expand}(l^*) = l) \Leftrightarrow (A(l^*) \subseteq A(l) \wedge (\forall\, a \in A(l^*)).(a(l^*) = a(l)))$$

Expansion means adding attributes and their values to the sighting's location record in order to make tests for equivalence (see above) possible. It relies on the implicit notion of ordering between location records (although this is not mentioned in [55]). Also, there appears to be the assumption that each location sighting is matched by at most one pre-defined location.

Expansion and equivalence form, implicitly, the basis of the algorithm. The following paragraphs describe the algorithm's functionality applying the stages of the acquisition stack identified earlier.

Reception This stage is concerned with querying sightings from sensors or low-level slave-locators. The low-level mappings to make sightings sensor-independent are performed by sensor-specific sub-systems (called slave locators).

Abstraction The abstraction stage is also performed by slave locators. It consists mainly of mapping sensor sightings into the attribute-based location model described above:

$$\mathbf{abstr}\colon (T \times L^{\tau}) \rightarrow L^*$$

$$\mathbf{abstr}\colon (\tau, l^{\tau}) \mapsto l^*$$

Effectively, this entails construction of a location record whose sensor identifier is stored as an attribute value.

Fusion The master locator queries its slave locators to collect their location records. The returned location records are *expanded* using the location directory. Subsequently, expanded sightings are fused using a *corroboration function*, which yields a *confidence weighted* priority queue of locations. The corroboration function uses the equivalence relation described above to test whether the locations returned by different slave locators are the same.

$$\textbf{fusion}: \{l_1^*, ..., l_n^*\} \mapsto (L \rightarrow C)$$

$$\textbf{fusion}(\{l_1^*, ..., l_n^*\}) = \textbf{corroborate}(\{\textbf{expand}(l_1^*), ..., \textbf{expand}(l_n^*)\})$$

The corroboration function uses hard-coded knowledge to arbitrate between conflicting sightings. More flexible policies are also mentioned, although no concrete details are given.

Discussion As indicated by the above description, it appears that the underlying location model was not clearly identified before designing this algorithm. While the attribute-based location representation is very powerful, it not a good formalism to define algorithms over location information. We believe that a formal location model would have facilitated the design of the acquisition algorithm. Further, the model could have been used to specify and document location processing independent of its implementation. The attribute-based location representation should have come into play only at the implementation level.

5.7.2 Translation into a lattice

In the previous section, we have argued that location–processing algorithms should be based on a formal location model. Hence, we shall use this section to sketch an acquisition algorithm based on an inclusion–ordering relation between locations.

Location model We use the symbolic model proposed in chapter 3: we treat locations as symbols, that is, opaque entities which can be tested for equivalence, inclusion, and overlap. To compare quality of sightings, we also require the location's geographical area.

Reception We propose to use sensor-type specific sub-system (e.g. an Active Badge service), or user-agents. (This is similar to Rizzo's algorithm.) Additionally, we require that each sighting be weighted with a confidence value expressing the probability that it is valid.

Abstraction The abstraction stage maps each sensor-type-specific location into a location symbol. This is achieved either by pre-defining location symbols for all sensor locations, or by creating location symbols dynamically.

$$\textbf{abstr}: l^\tau \mapsto l$$

As a result, a location symbol is associated with each sighting event.

Fusion This maps a set of location symbols $\mathrm{L}_{\mathbf{inp}}$ into a confidence function. We use the following algorithm over inclusion-ordered location lattices to fuse location data:

Firstly, we construct a lattice $\mathrm{L}_{\mathbf{lat}}$ from the input data that is closed against the greatest lower bound (*glb*) and the least upper bound (*lub*). For pairs of non-overlapping locations the greatest lower bound is not defined. Hence $\mathrm{L}_{\mathbf{lat}}$ can have multiple leaf nodes.

$$\begin{aligned} \mathrm{l} \in \mathrm{L}_{\mathbf{lat}} &\Leftarrow \mathrm{l} \in \mathrm{L}_{\mathbf{inp}} \\ \mathrm{l} \in \mathrm{L}_{\mathbf{lat}} &\Leftarrow (\exists \mathrm{l}1, \mathrm{l}2 \in \mathrm{L}_{\mathbf{inp}}).(\mathrm{l} = \mathit{glb}(\mathrm{l}_1, \mathrm{l}_2)) \wedge (\mathrm{l} \in \mathrm{L}) \\ \mathrm{l} \in \mathrm{L}_{\mathbf{lat}} &\Leftarrow (\exists \mathrm{l}1, \mathrm{l}2 \in \mathrm{L}_{\mathbf{inp}}).(\mathrm{l} = \mathit{lub}(\mathrm{l}_1, \mathrm{l}_2)) \wedge (\mathrm{l} \in \mathrm{L}) \end{aligned}$$

We construct the *smallest* set $\mathrm{L}_{\mathbf{lat}}$ satisfying these constraints. The bounds (*glb* and *lub*) are computed with respect to the spatial inclusion ordering, an asymmetric and transitive relationship.

Intuitively, the *glb* closure identifies overlaps, and includes them as separate symbolic locations.[2] The *lub* closure adds redundant lower-resolution locations. While this is not essential to the algorithm, it is convenient for subsequent multi-resolution processing and detail filtering. If the location hierarchy is static, the least upper bound closure can be deferred.

If the lattice $\mathrm{L}_{\mathbf{lat}}$ has more than one leaf node, the set of sightings refers to more than one physical location. Since we assume that a located-object can only be in one place at a given time, the set of sightings is then inconsistent. To remove these inconsistencies, two approaches are possible:

- Construct the biggest conflict-free subset of $\mathrm{L}_{\mathbf{lat}}$. This we term the *consensus* approach.
- Construct multiple conflict-free sets such that their union is equal to $\mathrm{L}_{\mathbf{lat}}$. This approach we call *factoring*.

Both approaches are built on the notion of a conflict-free lattice.

Property 6 *We call a lattice* $\mathrm{L}_{\mathbf{cf}}$ conflict-free *if, and only if, it satisfies the following condition:*

$$(\forall \mathrm{l}_1, \mathrm{l}_2 \in \mathrm{L}_{\mathbf{cf}}).(\exists \mathrm{l}_3 \in \mathrm{L}_{\mathbf{cf}}).(\mathrm{l}_3 = \mathit{glb}(\mathrm{l}_1, \mathrm{l}_2))$$

□

A corollary is that each $\mathrm{L}_{\mathbf{cf}}$ has at most one leaf node, that is, a *least element*.

Figures 5.5 and 5.6 illustrate the construction of $\mathrm{L}_{\mathbf{lat}}$. In Figure 5.6 there are three conflict-free lattices. Each has a least element that is spatially contained by all other elements.

[2] We assume that all defined leaf nodes in **L** are non-overlapping.

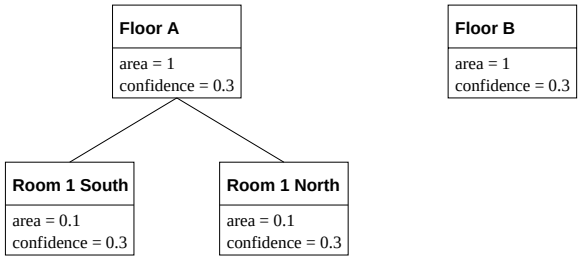


Figure 5.5: Set of location sightings before fusion

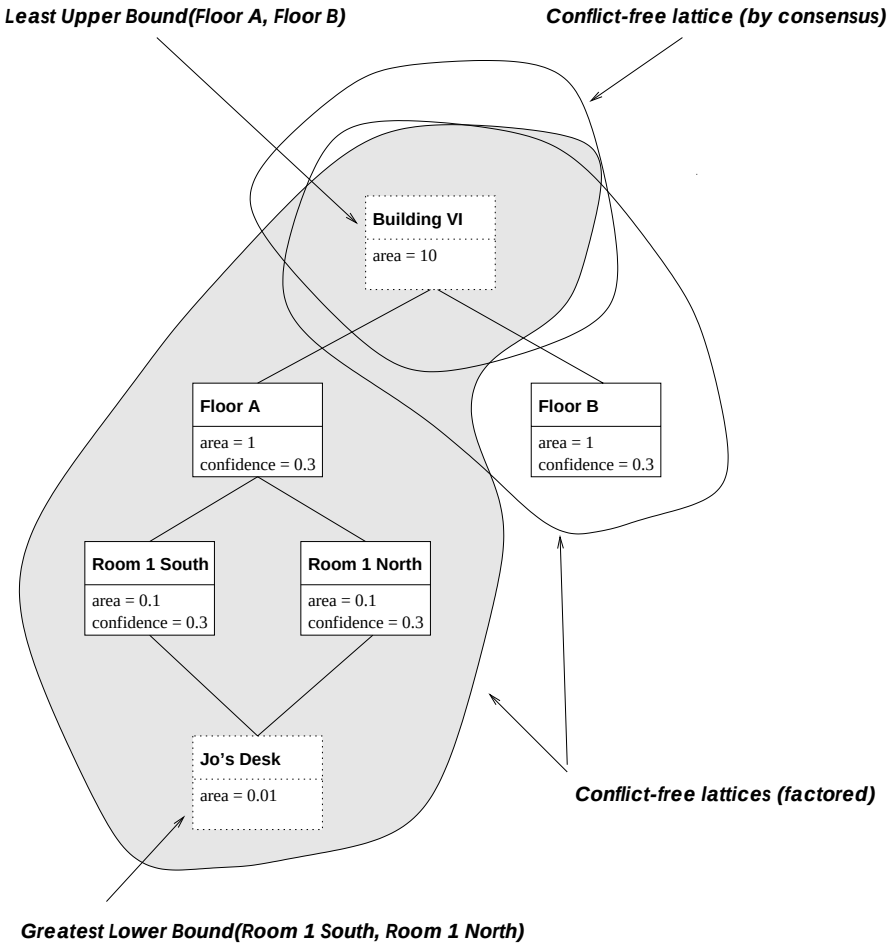


Figure 5.6: $L_{\mathbf{lat}}$ with conflict-free lattices

Consensus method With this approach, we aim to make $\mathrm{L}_{\mathbf{lat}}$ consistent by removing as few nodes as possible. Essentially, we remove all nodes from $\mathrm{L}_{\mathbf{lat}}$ that do not overlap at least one other location in $\mathrm{L}_{\mathbf{lat}}$ whose greatest lower bound is not defined. We construct the conflict-free lattice with the following rule:

$$\mathrm{l} \in \mathrm{L}_{\mathbf{cf}} \quad \Leftarrow \quad \mathrm{l} \in \mathrm{L}_{\mathbf{lat}} \wedge (\forall \mathrm{l}_1 \in \mathrm{L}_{\mathbf{lat}}).(\mathrm{l} < \mathbf{l}_1 \vee \mathrm{l}_1 < \mathbf{l} \vee \mathrm{l} = \mathrm{l}_1)$$

The $\mathrm{L}_{\mathbf{cf}}$ constructed by this method is guaranteed to be non-empty if the location hierarchy has a greatest element. Figure 5.6 shows the result of the consensus method.

Factoring An alternative (and possibly more useful) method is to factor $\mathrm{L}_{\mathbf{lat}}$ into multiple conflict free lattices. This has the advantage that location accuracy is preserved. We identify distinct leaves in $\mathrm{L}_{\mathbf{lat}}$ and construct a separate lattice $\mathrm{L}_{\mathbf{cf}}$ for each leaf such that all locations are greater or equal than the defining leaf.

$$\begin{aligned} \mathrm{l}_{\mathbf{leaf}} &\in \mathrm{L}_{\mathbf{cf}} \\ \mathrm{l} \in \mathrm{L}_{\mathbf{cf}} \quad &\Leftarrow \quad (\mathrm{l} \in \mathrm{L}_{\mathbf{lat}}) \wedge (\mathrm{l}_{\mathbf{leaf}} < \mathrm{l}) \end{aligned}$$

Since a lattice is inconsistent only if it has multiple leaves, this method constructs conflict-free lattices.

If $\mathrm{L}_{\mathbf{lat}}$ is inconsistent, the factoring algorithm will generate multiple non-conflicting lattices $\mathrm{L}_{\mathbf{cf}}$. Each represents an alternative location for a located-object. Therefore, we need to order those locations according to some metric of confidence or quality. For example, we could use:

$$\mathbf{metric}(\mathrm{L}_{\mathbf{cf}}) = \sum_{\mathrm{l} \in \mathrm{L}_{\mathbf{cf}}} \mathbf{confidence}(\mathrm{l}) \cdot (1 \Leftrightarrow \mathbf{area}(\mathrm{l}))$$

This is a heuristic metric combining sighting confidence with the size of sighting area. The rationale is that accuracy and validity are two sides of the same coin, and that applications may favour accuracy if the confidence values are roughly equivalent. (Other metrics are possible.) In any case, it is crucial to get an accurate indication of sighting validity from the sensors.[3]

Discussion By using a hierarchical location model, our acquisition algorithm is based on well-understood set-theoretical concepts. In particular, the following features of the hierarchic location model are exploited:

- We can determine whether a node is a refinement (i.e. sub-location) of another node. Refinement is indicative of one sighting supporting another.
- We can determine whether two nodes overlap. This is the case if they have common descendants. Non-overlapping nodes are indicative of conflicting sightings.

[3] In chapter 6 we discuss in greater detail measures of validity and their effect on location processing.

- All ancestors of each location are known. Hence, results can be returned and subsequently processed at multiple resolutions.

As a result, our algorithm can exploit overlapping sightings to increase accuracy. Further, multi-resolution sensor-systems and applications can be supported.

The proposed algorithm can be implemented by extending the attribute-based representation proposed by Rizzo *et al*. To do so, we use subset inclusion over attribute values to model spatial inclusion of locations. Thus, a parent location would contain a superset of the attribute values of all its children.

5.8 Case study: An active office system

In this section, we discuss an existing system implementing acquisition of location data. We examine an experimental "Active Office" location system that uses location information from Active Badges and UNIX workstations. It has been in operation for over two years in our research group.

An application is the `cwhere` client program. When invoked, it produces a list of people (denoted by their UNIX account name), along with their last known location and the time when they were seen there:

```
scorch.doc.ic.ac.uk% cwhere
  yt3 was at 336H  Ph.D. Room 48249      on Mon Oct 13 21:47:50
   az now in 563aH Ph.D. Room 58239      seen 23 seconds ago
  mpn was at 558cH Ph.D. Room 58238      on Thu Jul 31 15:19:08
   ul now in 336H  Ph.D. Room 48249      seen 4 seconds ago
 ndcm was at 558cH Ph.D. Room 58238      on Thu Oct 16 17:37:23
 jcf1 was at 557H  Room 557              on Thu Oct 16 23:26:29
  gu1 now in 5xxH  Level 5               seen 5 seconds ago
  ar3 was at 359H  A. Russo 48353        on Thu Oct 16 18:54:42
  ban was at 428H  Bashar Nuseibeh 48286 on Thu Oct 16 20:05:59
  wlp was at 563aH Ph.D. Room 58239      on Thu Oct 16 20:45:24
  vh3 was at 564H  Vassos 48355          on Mon Aug 18 17:21:41
 paul was at 429Ha DSE-Lab (NE) 48293    on Fri Oct 17 04:08:34
scorch.doc.ic.ac.uk%
```

The system architecture is shown in Figure 5.7. There are two location services used by the `cwhere` client: an Active Badge service (as implemented by Jeff Magee [41]), and a UNIX location service based on `ruser` daemons. As support platform we use `Regis`, a lightweight multi-threaded distributed programming environment [41].

The badge location service is provided by the `Regis` component `badgeman`, connected via TCP/IP to the `abpoll` daemons. Each `abpoll` drives a chain of badge sensors via an RS.232 serial link. Whenever a badge sensor receives a message from a badge, it is relayed through an `abpoll` process to `blocate`.

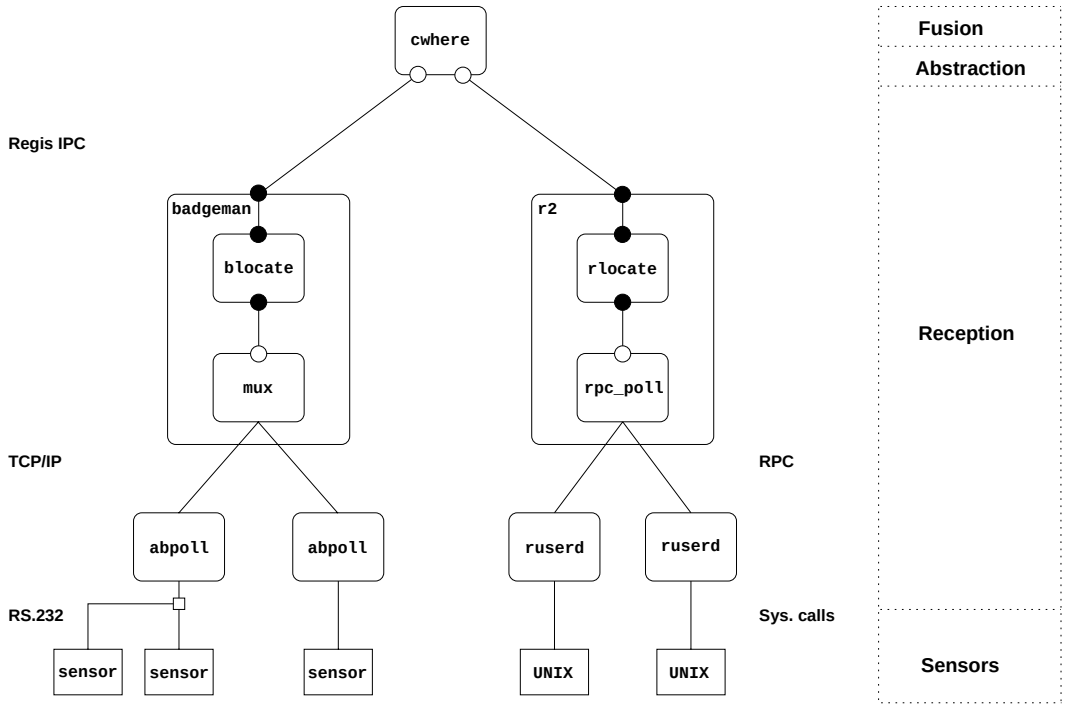


Figure 5.7: Active office location system

The `blocate` component of `badgeman` is responsible for collecting badge sightings into a badge-indexed database, which is stored persistently. Clients query `blocate` via the interface exported through `badgeman` for the current location of users. Clients can also subscribe to location events. All client interactions use badge identifiers and badge sensor identifiers to refer to users and locations, respectively.

The UNIX location service mirrors the structure of the badge location service. The main difference is that the sensors do not generate location events (as badge sensors do), but need to be queried. Hence, the `rpc_poll` component periodically queries the `ruser` daemons via RPC calls. In turn, the `ruser` daemons read the local `/etc/utmp` file to find out who is logged on. By calling kernel functions the idle time for each login session is established.

Similarly to `blocate`, `rlocate` maintains a persistent database of current user locations. This time, UNIX login names and DNS host names are used as representation schemes for users and locations, respectively. The `rlocate` database can be queried for users' current locations. Also location events are available.

The client, `cwhere`, connects to both `badgeman` and `r2` to query all location entries which are not older than a certain threshold. After unifying the representation schemes, entries from both systems are merged, and the result of this process displayed.

Reception In the Active Badge subsystem, the functional mappings of the reception layer are performed inside `mux`. For each sighting, it prefixes the badge sensor identifier (which is unique per sensor chain) with the identifier of the sensor chain. The UNIX

location subsystem does not perform mappings since UNIX account names and DNS host names are unique in the context of the subsystem.

Abstraction Abstraction mappings are performed inside the `cwhere` client. To integrate user identifiers, we chose to use UNIX account names as the unified representation domain. The motivation was that while only a few people have badges, all have a UNIX account (at least in our Department). For location, we chose badge sensor identifiers as the abstract representation domain. While this limits the accuracy and coverage of the system, it allows the reuse of the pre-existing location directory of the Active Badge system.

Fusion The fusion algorithm is applied to sets of sightings that are related to the same user. In our case, such a set consists at most of one badge sighting and one UNIX sighting. The algorithm consists of two phases: conflict detection and conflict removal. To detect a conflict between two sightings, we check whether both correspond to the same location. If not, we compare the timestamps of the sightings and choose the more recent sighting. In case the timestamps are equal, we use the badge sighting. By using these two rules, each of the sets is reduced to a single "current" sighting. Note that fusion of sightings is a more general problem than fusion of locations discussed earlier in this chapter.

Discussion

The described system uses a special-purpose acquisition function. Abstraction and Fusion are tailored to the processing of badge sightings and UNIX location sightings. We have applied the acquisition stack to understand the location processing in the system. This is remarkable since the system was not implemented as a design exercise, but in order to get a working user tracking system.

Architecturally, it is arguable whether client side abstraction and fusion is the best choice. It certainly leads to simpler server implementations, which in turn makes servers more light-weight and more robust. However, reuse of the processing logic is easier when exposed as a service. Further, client-side processing in this case substantially increases the bandwidth requirement.

5.9 A note on acquisition in robotics

Acquisition and fusion of sensor data is a classic problem in robotics. A typical robot has multiple sensors (sound sensors, vision sensors, etc.) which let it monitor different aspects of its environment. The robot needs to combine the inputs from these sensors in order to gain a more complete knowledge of its environment.

For example, Alberto Elfes has designed a sonar-based navigation system for robots [16]. The system constructs a picture of the robot's environment from the data collected by an array of sonar sensors. Subsequently, the robot matches feature descriptions from a database against the combined sensor input. As a result, the robot can determine its own physical location.

The system uses a layered architecture. It includes the following layers:[4]

- A sensor-interpretation layer acquires and interprets sensor data.
- A sensor-integration layer translates the data into an abstract common format.
- A real-world-modelling layer integrates the data into a global world model. Object-recognition and map matching are used as heuristic integration methods.

This layering approach bears resemblance to our approach since it is driven by the need to combine multiple sensor systems. However, in robotics sensor fusion is used in order to *recognise* or *match* features of the environment (objects, landmarks, etc.). The result is context-awareness, rather than location-awareness. Further, the algorithms are often numerical (i.e. focussed on "number crunching"). Hence, we have found that robotics-related research is quite far removed from our work.

5.10 Chapter Summary

The collection of data from location sensors, their subsequent abstraction and fusion are performed by the acquisition function. In this chapter, we have identified the functional and structural requirements that need to be addressed by the acquisition function of a location service. The main functional requirements are generality and adequate spatio-temporal resolution. The important structural requirements include openness and real-time information delivery.

After a discussion of architectural design issues, we have proposed the acquisition stack as the basic structure of the acquisition function. The stack consists of reception, abstraction, and fusion layers. We have presented a function-based model for layers and their interaction.

We have examined a concrete fusion algorithm in order to demonstrate that a formal understanding of the location model facilitates the design of an acquisition algorithm. We have proposed a new algorithm based on the ordering of locations by spatial inclusion. The algorithm allows for overlapping locations and multi-resolution input. Inconsistencies in the location input are dealt with either by finding the maximum consensus or by factoring the input into conflict-free sets.

Finally, we have examined an existing location service and applied the abstract structure described earlier in this chapter. We have found that the elements of the acquisition stack can be identified, and their identification helps to structure and understand the system.

[4]The more robotics-oriented higher layers have been omitted.

Chapter 6

Uncertainty, Prediction, and Interpolation

Location sensors provide location information at discrete points in time in the past. Often, these points in time are determined by availability of location information rather than demand for it. This lack of synchronisation between supply and demand, and physical limitations of location sensors, are the main reasons why location information is *always* incomplete, and *sometimes* incorrect. This property we call **uncertainty** of location information.

Typical context-aware applications use location data to adjust their present behaviour and to prepare for the immediate future (e.g. by pre-allocating resources and pre-fetching data). However, location tracking systems can only deliver information about the past (even if this includes directions or intention for future moves). Location **prediction** helps deduce future and present locations from past location data.

Missing information, incorrect information, and prediction may cause relatively short temporal gaps in the availability of location information. While location-awareness is often event-driven (and thus unaffected by gaps in availability), there will often be a complementary requirement for location information for times when it is or was not available. Thus, there is a need to bridge temporal gaps in the available location information, a process we refer to as **interpolation**.

Interpolation and prediction reduce uncertainty by exploiting mobility models, that is, knowledge of how located-objects move through space. Uncertainty is thus reduced, but is never eliminated completely. Hence, it needs to be tolerated at all processing stages.

Prediction and interpolation are performed by the Trace function (cf. Figure 4.1, page 72). While this chapter does not offer a comprehensive solution for this layer, we identify the available technologies and abstractions. Further, we propose metric-based selection of heuristics as a general mechanism to reduce uncertainty.

In the remainder of this chapter, we discuss causes and models of uncertainty. We propose metrics that allow us to measure completeness and correctness of individual events and traces of related events. Further, we discuss extant mobility models and their applicability to certain classes of correctness and completeness. Finally, we describe interpolation across events and state traces in order to integrate predicted locations.

6.1 Preliminaries

We use the hierarchical location model described in chapter 3 in which each symbolic location $\mathbf{l}$ has a well-defined area of size $\mathrm{A}(\mathbf{l})$. The ordering of symbolic locations represents the spatial *contains*-relation.

6.1.1 Sightings and traces

A *sighting event* $\mathbf{s}$ is described by a tuple

$$\mathbf{s} = (\mathbf{t}, \mathbf{l}, \mathbf{p})$$

where $\mathbf{t^s}$ is the time of the sighting, $\mathbf{l^s}$ is the symbolic location of the sighting , and $\mathbf{p^s}$ is the probability that the sighting is valid. This corresponds to the information provided by the acquisition layer and stored by the tracking function.

For each located-object, sighting events $\mathbf{s_i}$ are composed into a trace $\mathbf{T}$, a time-ordered set with start-time $\mathbf{t_s^T}$ and end-time $\mathbf{t_e^T}$.

$$\mathbf{T} = (\mathbf{t_s}, \mathbf{t_e}, \langle \mathbf{s}_1, ..., \mathbf{s_n} \rangle) \text{ with } \mathbf{t_s^T} \leq \mathbf{t^{s_1}} \text{ and } \mathbf{t^{s_i}} < \mathbf{t^{s_{i+1}}} \text{ and } \mathbf{t^{s_n}} \leq \mathbf{t_e^T}$$

We shall assume, with no loss of generality, that no two events are simultaneous. Thus, exactly one sequence of sighting events can be constructed.

Besides the notion of a trace of discrete events, we define an interval $\mathbf{I}$ as the time for which an object remains at a given location $\mathbf{l}$.

$$\mathbf{I} = (\mathbf{t_s}, \mathbf{t_e}, \mathbf{l}) \text{ with } \mathbf{t_s^I} \leq \mathbf{t_e^I}$$

Further, we define an interval trace as a sequence of intervals, each corresponding to a location, and there may be gaps but no overlaps between the intervals.

$$\mathbf{IT} = (\mathbf{t_s}, \mathbf{t_e}, \langle \mathbf{I}_1, \mathbf{I}_2, ..., \mathbf{I_n} \rangle) \text{ where } \mathbf{t_s^{IT}} \leq \mathbf{t_s^{I_1}} \text{ and } \mathbf{t_i^e} \leq \mathbf{t_{i+1}^s} \text{ and } \mathbf{t_e^{I_n}} \leq \mathbf{t_e^{IT}}$$

Continuous interval traces are state traces, or state transition flows, a special case of interval traces:

$$\mathbf{ST} = (\mathbf{t_s}, \mathbf{t_e}, \langle \mathbf{I}_1, \mathbf{I}_2, ..., \mathbf{I_n} \rangle) \text{ where } \mathbf{t_s^{ST}} = \mathbf{t_s^{I_1}} \text{ and } \mathbf{t_i^e} = \mathbf{t_{i+1}^s} \text{ and } \mathbf{t_e^{I_n}} = \mathbf{t_e^{ST}}$$

Note that the definitions of interval traces and state traces are only useful over a *disjoint* set of locations. Otherwise, a located-object can be in more than one location at a time. We address this issue using the concept of a movement plane.

6.1.2 Movement planes

A movement plane $\mathbf{M}$ is a tuple $(\mathbf{L}, \mathit{Adj}, \mathit{Dist})$ with

$$\begin{aligned} \mathbf{L} &: \mathbb{P}\,\mathbf{LOC} \\ \mathit{Adj} &: \mathbf{LOC} \leftrightarrow \mathbf{LOC} \\ \mathit{Dist} &: \mathbf{LOC} \times \mathbf{LOC} \rightarrow \mathbf{DIST} \times \mathbf{DIST} \\ \mathit{Dir} &: \mathbf{LOC} \times \mathbf{LOC} \rightarrow \mathbf{DIR} \times \mathbf{DIR} \end{aligned}$$

$\mathbf{L}$ is the set of non-overlapping locations that constitutes the spatial coverage of the movement plane $\mathbf{M}$.

The binary relationship *Adj* relates physically adjacent locations. If $\mathbf{A}$ is adjacent to $\mathbf{B}$, then a located-object can travel *directly* from $\mathbf{A}$ to $\mathbf{B}$. Hence, *Adj* is dependent on the capabilities of the located-object — what is adjacent for a seagull may not be for an ostrich. Thus a given movement plane is only valid for located-objects satisfying certain conditions. Adjacency is directional (i.e. not symmetric).

The distance function *Dist* provides a quantification of distance between locations. Since each location covers an area of non-zero size, *Dist* yields two values: a lower bound $\mathbf{d}_{min}$ and an upper bound $\mathbf{d}_{max}$.

$$Dist(\mathbf{A}, \mathbf{B}) = (\mathbf{d}_{min}, \mathbf{d}_{max})$$

If one located-object is situated at $\mathbf{A}$ and another located-object at $\mathbf{B}$, the distance between those object is guaranteed to be between $\mathbf{d}_{min}$ and $\mathbf{d}_{max}$.

Analogously, the directionality function *Dir* maps pairs of locations to minimum and maximum bearings between objects in the respective locations. Effectively, *Dir* defines the notion of a straight line within a Movement Plane. Located-objects are inert and thus follow straight lines unless redirected by some event. Note that "inertia" and "straight line" could mean different things for different kinds of located-objects.

In order to relate incoming location information to a selected movement plane, it is necessary to restrict an event trace $\mathbf{T}$ to a movement plane $\mathbf{M}$:

$$\mathbf{T} \setminus \mathbf{M} = (\mathbf{t}_\mathbf{s}^\mathbf{T}, \mathbf{t}_\mathbf{e}^\mathbf{T}, \left\langle \mathbf{s}_1^\mathbf{T}, ..., \mathbf{s}_\mathbf{n}^\mathbf{T} \right\rangle \setminus \mathbf{L}^\mathbf{M})$$

By restricting $\mathbf{T}$ to $\mathbf{M}$, all spatially irrelevant sightings are removed. Also, the locations of remaining sightings are matched, if possible, against the locations in $\mathbf{M}$.

Taking advantage of the hierarchical nature of location space, we define *restriction* of a sequence of sightings to a set of locations as follows:

$$\langle\rangle \setminus \mathbf{L} = \langle\rangle \tag{6.1}$$

$$(\langle \mathbf{S} \rangle : \mathbf{s}) \setminus \mathbf{L} = \begin{cases} \mathbf{l}^\mathbf{s} \in \mathbf{L} & \langle \mathbf{S} \rangle \setminus \mathbf{L} : \mathbf{s} \\ (\exists \mathbf{l} \in \mathbf{L}).(\mathbf{l} < \mathbf{l}^\mathbf{s}) & \langle \mathbf{S} \rangle \setminus \mathbf{L} : (\mathbf{t}^\mathbf{s}, \mathbf{l}, \mathbf{p}^\mathbf{s}) \\ \textbf{otherwise} & \langle \mathbf{S} \rangle \setminus \mathbf{L} \end{cases} \tag{6.2}$$

The second case of 6.2 includes events that originally have a higher spatial resolution. This mapping is unique since the members of $\mathbf{L}$ are non-overlapping. Restriction over interval traces is defined analogously.

Note: sightings of lower spatial resolution, which may be spatially relevant, are discarded. To extract them a lower-resolution movement plane must be used.

6.2 Uncertainty

Location information is potentially deficient in two ways: it can be incomplete (lack of precision) and it can be incorrect (lack of accuracy). We subdivide these two general

notions into more measurable sub-categories:

- Elements of *Correctness*: spatial accuracy and temporal accuracy, validity.
- Elements of *Completeness*: spatial and temporal sampling resolution.

They are related. Correctness typically imposes an upper bound on achievable completeness given a location sensor technology. Sensor systems often have inherent upper limits of correctness and completeness.

Correctness and completeness ultimately indicate how *faithfully* location information reflects movement of located-objects. Therefore, *absolute* measures of correctness and completeness require some knowledge of their mobility (such as maximum speed or size). If such knowledge is not available, *comparative* metrics over sets of objects with similar mobility must be employed.

Spatial accuracy is a measure of *distance* between actual and reported location. With coordinate-oriented sensors its measurement is straightforward. Some sensor systems (such as GPS) can dynamically determine the accuracy of their measurements, others may have known accuracy limits. Cell-oriented sensors are based on a well-defined level of accuracy, hence the size of the cell already reflects potential spatial inaccuracy in the sighting.

Temporal accuracy is a measure of *time* between actual and reported sighting. Temporal inaccuracies are caused by clock drift, transmission delays, and other physical limitations of the tracking system. Also the acquisition stack can introduce temporal inaccuracies. It should be noted that GPS sensors can measure the time of a sighting very accurately, while others have no notion of time whatsoever (such as Active Badge sensors).

Validity is a qualitative measure of a sighting's truth. An invalid sighting differs from the truth by an *unknown* amount. This is the case, for example, if a location tag has been separated from its bearer. Similarly, false authentication of located-objects can lead to invalid location information. We use *confidence values* to represent the probability of a sighting being valid.

Spatial sampling resolution determines the spatial *precision* of a sighting. The accuracy of location sensors imposes an upper bound on the precision of location sightings. For instance, the cell size in GSM determines the precision of the location information in that system. However, it is possible to reduce the spatial sampling resolution arbitrarily (e.g. to reduce update volumes).

Temporal sampling resolution is the frequency with which a located-object's location is sampled. In relation to a located-object's maximum rate of location change, this indicates how closely the movements of that object can be monitored. As implied by Nyquist's Sampling Theorem, a given sampling rate can only faithfully monitor a certain maximum rate of location change. Thus, any temporal sampling resolution will only be adequate for located-objects below a certain speed. The maximum possible sampling

resolution is limited by resource constraints (power, bandwidth). There is a correlation between spatial and temporal sampling resolution: smaller locations imply a higher rate of location changes, which in turn require a higher sampling frequency. Also, social factors place an upper bound on the frequency of monitoring people's movements.

Discussion

In order to reduce complexity, it is desirable to reduce the number of uncertainty factors which need to be considered. Spatial accuracy can be accommodated by adjusted precision. Temporal inaccuracy can safely be ignored as long sensor systems are not applied outside their design envelope (i.e. for located-objects that move too fast for them).

Temporal and spatial resolution can be determined easily. Using knowledge of the mobility of a given located-object, both can be composed, along with the confidence measures, into a probabilistic location space (see below).

6.2.1 Representing uncertainty

The fundamental requirement of an uncertainty model is to give an indication of the *likelihood* that a given located-object was, is, or will be, at a particular location at a given time. Thus, we model the space of location uncertainty as a function $\mathbf{f}$:

$$\mathbf{f} : \mathbf{LOC} \times \mathbf{T} \rightarrow [0, 1]$$

This function maps a symbolic location to a probability that the object is there at a given time. For a single instant $\mathbf{t}$ in time, this simplifies to:

$$\mathbf{f^t} : \mathbf{LOC} \rightarrow [0, 1]$$

Such a function has already been used (chapter 5) as a confidence function to model uncertainty in the output of the acquisition stack. $\mathbf{f^t}$ can be represented as a set of pairs $(\mathbf{l_i}, \mathbf{p_i})$ of locations and probabilities. This facilitates the pruning of pairs with low probabilities or with locations outside a particular area. Such pruning is necessary in order not to to swamp location-aware applications with location data irrelevant to them. We believe that a single pair $(\mathbf{l_i}, \mathbf{p_i})$ will often be sufficient.

6.2.2 Rationale

Figure 6.1 shows how we model incompleteness and incorrectness in location information. The spatial and temporal *sampling resolution* is a systematic source of incompleteness, whereas *noise* leads to both incorrectness and additional incompleteness. We distinguish those two sources of uncertainty because the sampling resolution is, in principle, known in advance, whereas noise occurs non-deterministically.

Between the actual movements of the located-objects (Reality) and the gathered location data (Actual Sample), we introduce the abstraction of a *Perfect Sample*, the maximum amount of data that could have been gathered given the movements of the located-object and the available location sensors. Hence, it contains less information than

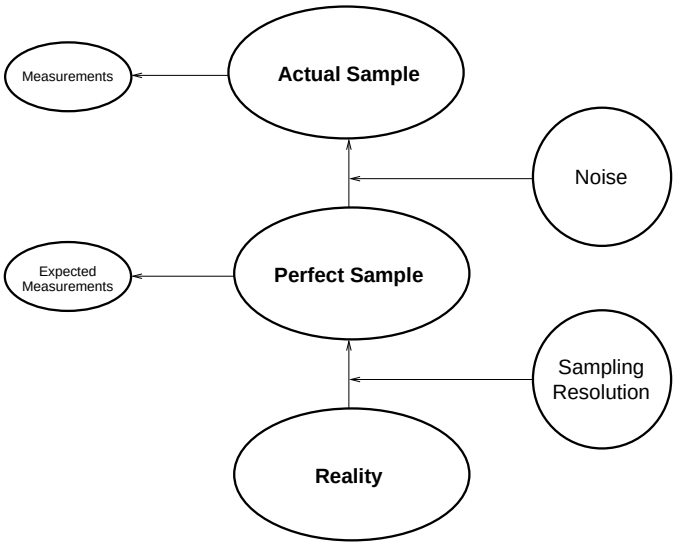


Figure 6.1: Model of uncertainty in location information

is inherent in the actual movement, the reduction being determined by the spatial and temporal sampling resolution.

An actual sample is a noise-degraded perfect sample. Hence, the perfect sample marks the upper bound of the actual sample with respect to correctness and completeness of information. Noise may manifest itself as lost, phantom, or distorted sightings. It is a random occurrence caused by external influences. Such a non-systematic degradation of location sightings can only be determined by measurements.

Despite its deterministic nature, in a large and heterogeneous environment the spatio-temporal sampling resolution can be a product of many factors, including the capabilities and the location of the located-object. In such cases, it may be easier to measure the spatio-temporal sampling resolution rather than estimate it. However, if the sampling resolution is not known in advance, it is impossible to distinguish the effect of noise from the effect of the sampling resolution. Ultimately, it does not really matter whether a sighting is not available because it was lost due to noise or because the sensor system was not able to acquire it. Hence, for measuring uncertainty we do not necessarily rely on knowledge of the characteristics of the perfect sample. We compute measurements over the actual sample, which may be compared against expected measures if available.

6.2.3 Measuring uncertainty

While we assume that location sightings enter the system tagged with some confidence value (measured by sensors or calculated through error analysis), we need ways to extract more abstract measures from this data. It is especially necessary to establish the amount of uncertainty in our knowledge of the location trace of a particular object. This naturally requires some compounded uncertainty metrics. Such compounded measures are intended to be used by later processing stages to select appropriate heuristics, since each prediction method tends to be most effective if the input trace has certain characteristics. Further,

the quality of a prediction is bound to depend on the trace information it was based on.

Typical deficiencies of sighting traces include lack of sightings during prolonged periods of time, and imprecise (i.e. too coarse-grained) locations. A compounded metric must facilitate the detection of those shortcomings.

In the following, we propose a series of measures that can be computed for sightings and traces to give an indication of the quality of the available information.

Quality measures for single sighting events

The two main dimensions of quality for single sighting events are spatial resolution and reliability.

Firstly, each sighting $\mathbf{s}$ is tagged with a confidence value $\mathbf{p^s}$, i.e. a **probability of validity**. This is a good indication of whether a sighting is correct. Typically, a sighting's validity-probability is derived from physical characteristics of the tracking system, and perhaps the user profile (some users might give their Active Badge away quite often).

Secondly, from the sighting's location we can derive the sighting's **relative area**:

$$\mathrm{A_r}(\mathbf{s}) = \frac{\mathrm{A}(\mathbf{l^s})}{\mathrm{A}(anyLoc)} \text{ where } (\forall\, \mathbf{s}).(\mathrm{A_r}(\mathbf{s}) \in [0,1])$$

$\mathrm{A_r}(\mathbf{l^s})$ is the relative coverage of a location relative to the 'Root' location *anyLoc* (page 51) of the location hierarchy. A sighting's relative area is a measure of spatial precision, and implicitly a measure of accuracy. A value of *zero* indicates the highest possible precision: a point-sized area. On the other hand, a value of *one* means that the sighting is void of any spatial information.

Intuitively, the amount of information conveyed by a single sighting depends both on the sighting's precision and the probability that the sighting is valid (i.e. accurate).[1] To capture this duality of precision and confidence, we define the energy (or entropy) $\mathrm{E}(\mathbf{s})$ of a sighting $\mathbf{s}$ as:

$$\mathrm{E}(\mathbf{s}) = \mathbf{p^s}(1 \Leftrightarrow \mathrm{A_r}(\mathbf{s})) \text{ with } \mathrm{E}(\mathbf{s}) \in [0, \mathbf{p^s}]$$

A sighting's **energy** weights the "payload" of information that is carried by the sighting. The larger the relative area of the location, the less newsworthy it becomes. Conversely, the probability of validity is correlated with the amount of uncertainty removed by the sighting.

We shall use the energy as the main measure of uncertainty because it captures both correctness and completeness. Thus, it is a measure of the sighting's quality. However, it characterises only one sighting at a time. In the next section, we discuss how measures over sets of related sightings can be obtained.

Trace metrics

When considering a set of location events from an application's point of view, two criteria are important. Firstly, information has to be available. Secondly, available information has to be of a certain quality, that is, of a minimum spatial resolution and reliability.

---

[1] Information is often described as *reduction of uncertainty*.

A set of sightings (i.e. trace) $\mathbf{T}$ is characterised by the number $\mathbf{n^T}$ of events it contains. Further, the *duration* $\Delta\,\mathbf{t^T}$ of the trace $\mathbf{T}$ is defined as:

$$\Delta\,\mathbf{t^T} = \mathbf{t_e^T} \Leftrightarrow \mathbf{t_s^T}$$

These two simple trace measures are prerequisites for availability and quality metrics.

Availability The overall availability of location sightings is characterised by the event density: the number of events occurring per unit of time. The event density is closely related to the average time between events, which is not discussed here.

The **temporal sighting density** $\delta_\mathbf{n}(\mathbf{T})$ measures the average sighting density over the duration of trace $\mathbf{T}$:

$$\delta_\mathbf{n}(\mathbf{T}) = \frac{\mathbf{n^T}}{\Delta\,\mathbf{t^T}}$$

This metric gives an indication of the temporal resolution of the information available about a given located-object. However, it does not distinguish recent sightings from old ones, and does not take account of the spatial resolution, or reliability, of sightings.

Often, the availability of *recent* information is of greater interest than an overall average. Such a bias in favour of recent sightings can be modelled by an *aging function* weighting events based on their age. We propose $e^\mathbf{x}$ as the basis for the aging function:

$$age^\mathbf{T}(\mathbf{s}) = e^{-\mathbf{a}(\mathbf{t_e^T} - \mathbf{t^s})}$$

This function converges to 0 with age of sighting. For the most recent sightings it yields 1. The steepness of the aging function is tuned by the constant $\mathbf{a}$. With big $\mathbf{a}$, all but the most recent sightings will effectively be filtered out. Conversely, small $\mathbf{a}$ results in slow aging of sightings. In the degenerate case when $\mathbf{a} = 0$, all sightings are equal regardless of age. In effect, $\mathbf{a}$ lets us define the notion of an event's half-life:

$$\mathbf{t_{half}} = \frac{\ln 2}{\mathbf{a}}$$

After $\mathbf{t_{half}}$, an event has lost half its weight.

Thus, an aged-biased availability measure can be defined. The **aged sighting density** computes a measure of temporal sighting density that favour recent sightings:

$$\widetilde{\delta_\mathbf{n}}(\mathbf{T}) = \sum_{\mathbf{s}\in\mathbf{T}} age^\mathbf{T}(\mathbf{s}) \tag{6.3}$$

$$= \sum_{\mathbf{s}\in\mathbf{T}} e^{-\mathbf{a}(\mathbf{t_e^T} - \mathbf{t^s})} \tag{6.4}$$

$$\widetilde{\delta_\mathbf{n}}(\mathbf{T}) = e^{-\mathbf{a}\cdot\mathbf{t_e^T}} \cdot \sum_{\mathbf{s}\in\mathbf{T}} e^{\mathbf{a}\cdot\mathbf{t^s}} \tag{6.5}$$

Computed over a particular trace, this measure gives an indication of availability of sightings at the end time $\mathbf{t_e}$ of the trace. If an event is old, its impact on the total will be small. If the trace is actually a stream of real-time location events, the aged-sighting density is a metric of *current* availability of location sightings. The transforma-

tion (equation 6.5) shows how the measure can be computed incrementally over a stream of incoming events.

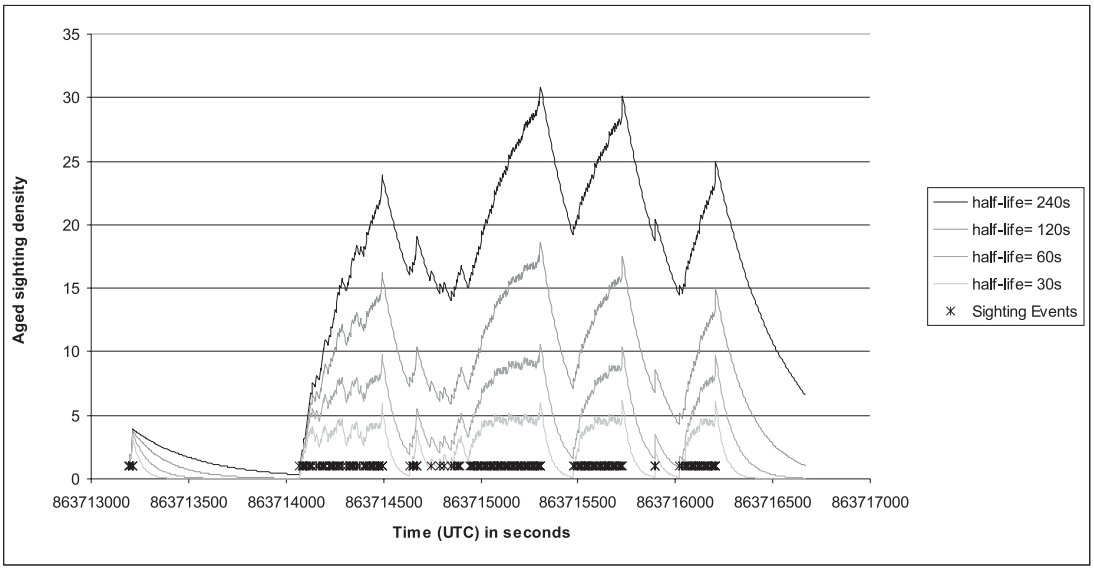


Figure 6.2: Aged sighting density for an Active Badge trace

Figure 6.2 shows the aged sighting density measure computed over a trace of sightings of a particular Active Badge.[2] For different event half-lifes, the measures converge to different ceilings. Given the values for **a** and the minimum time $min(\Delta\,\mathbf{t})$ between sighting events, we can derive the upper bound $X_{\delta_\mathbf{n}}$ as:

$$\begin{aligned} X_{\delta_\mathbf{n}}(min(\Delta\,\mathbf{t})) &= \lim_{\mathbf{n}\to\infty} \sum_{\mathbf{i}=0}^{\mathbf{n}} e^{-\mathbf{a}\cdot\mathbf{n}\cdot min(\Delta\,\mathbf{t})} \\ &= \frac{1}{1 \Leftrightarrow e^{-\mathbf{a}\cdot min(\Delta\,\mathbf{t})}} \end{aligned}$$

An Active Badge normally emits a beacon signal every 9 seconds. Therefore, the measures plotted in Figure 6.2 converge against the following ceilings in periods when no sightings are lost:

a	half-life	$X_{\delta_\mathbf{n}}(9)$
0.023104906	30	5.326299674
0.011552453	60	10.12662972
0.005776227	120	19.74026585
0.002888113	240	38.97403382

Note that the rate of convergence depends on the value of **a**. This is not surprising since **a** specifies the weight carried by older sightings. Effectively, **a** determines the metric's memory.

[2]The trace was generated by the "Active Office" system described in § 5.8.

The ceiling $X_{\delta_\mathbf{n}}$ characterises the maximum temporal sampling resolution. Hence, it is a measure of the temporal resolution of a perfect sample. The divergence of the actual $\delta_\mathbf{n}$ from $X_{\delta_\mathbf{n}}$ is due to noise, in particular due to lost sightings (assuming that the located-object has not ventured outside the coverage area of the Active Badge system).

On the other hand, the possibility that the located-object could have been in a room without an Active Badge sensor underlines the difficulty of establishing spatio-temporal resolution in the presence of noise. When the located-object was outside the reach of the sensors, the ceiling on the sighting density should have been adjusted accordingly. However, such adjustment requires knowledge of the object's location. Hence we conclude that $X_{\delta_\mathbf{n}}$ can only be an approximate measure of the perfect sample. Further, this indicates that the perfect sample can only be computed accurately if there is complete information about the located-object's movements. If such complete information is not available (i.e. all our knowledge is in the movement trace), we may not be able to distinguish the effects of noise from the effects of a temporarily lowered sampling resolution.

Quality Based on quality metrics for single events, the quality of a trace is characterised by the following averaged measures:

- Average area
$$\overline{\mathrm{A}}(\mathbf{T}) = \frac{1}{\mathbf{n^T}} \cdot \sum_{\mathbf{s}\in\mathbf{T}} \mathrm{A_r}(\mathbf{s})$$
- Average confidence
$$\overline{\mathrm{p}}(\mathbf{T}) = \frac{1}{\mathbf{n^T}} \cdot \sum_{\mathbf{s}\in\mathbf{T}} \mathrm{p^s}$$
- Average energy
$$\overline{\mathrm{E}}(\mathbf{T}) = \frac{1}{\mathbf{n^T}} \cdot \sum_{\mathbf{s}\in\mathbf{T}} \mathrm{E}(\mathbf{s})$$

These measures characterise the average quality of the sighting events within a trace. In addition, other statistical measures can easily be applied: median, variance, mean deviation, etc.

In order to measure the quality of recent sightings we calculate age-weighted averages. The age-bias is modelled by the same aging function as used earlier in this section.

- Age-averaged area
$$\widetilde{\mathrm{A}}(\mathbf{T}) = \frac{\sum_{\mathbf{s}\in\mathbf{T}}(\mathrm{A_r} \cdot age^{\mathbf{T}}(\mathbf{s}))}{\sum_{\mathbf{s}\in\mathbf{T}} age^{\mathbf{T}}(\mathbf{s})}$$
- Age-averaged confidence
$$\widetilde{\mathrm{p}}(\mathbf{T}) = \frac{\sum_{\mathbf{s}\in\mathbf{T}}(\mathrm{p^s} \cdot age^{\mathbf{T}}(\mathbf{s}))}{\sum_{\mathbf{s}\in\mathbf{T}} age^{\mathbf{T}}(\mathbf{s})}$$
- Age-averaged energy
$$\widetilde{\mathrm{E}}(\mathbf{T}) = \frac{\sum_{\mathbf{s}\in\mathbf{T}}(\mathrm{E}(\mathbf{s}) \cdot age^{\mathbf{T}}(\mathbf{s}))}{\sum_{\mathbf{s}\in\mathbf{T}} age^{\mathbf{T}}(\mathbf{s})}$$

These age-weighted averages can be used interchangeably with the "normal" averages, the only difference being that more recent sightings have a greater impact on the overall result.

These quality measures are useful only if there are significant variations in reliability or spatial resolution. This is often the case in a heterogeneous environment. In homogeneous location systems, such as Active Badge systems, all sightings are of similar quality with similar spatial accuracy and reliability (if the acquisition system is not too sophisticated). In such cases, only availability measures are of relevance.

Information density While it is useful to separate availability and quality metrics, in certain cases a single metric incorporating both is required. For example, we can envisage an application in which an alert is raised if somebody cannot be located anymore. The alert would be triggered either if no sightings were available, or available sightings were too coarse-grained.

The amount of information carried by a sighting $\mathbf{s}$ is measured by the sighting's energy $\mathrm{E}(\mathbf{s})$. Hence, we compute the **energy density** (or temperature) generated by the events of a given trace:

$$\delta_{\mathrm{E}}(\mathbf{T}) = \frac{\sum_{\mathbf{s}\in\mathbf{T}} \mathrm{E}(\mathbf{s})}{\Delta\, \mathbf{t}^{\mathbf{T}}}$$

This allows us to distinguish "red-hot" traces with detailed sightings and frequent updates from "cold" traces that receive only occasional low-quality sightings.

Analogous to the aged sighting density, we define **aged energy density** as:

$$\widetilde{\delta}_{\mathrm{E}}(\mathbf{T}) = \sum_{\mathbf{s}\in\mathbf{T}} (\mathrm{E}(\mathbf{s}) \cdot age^{\mathbf{T}}(\mathbf{s}))$$

The measure has the same age-bias characteristic as the aged sighting density. If there is no variation in sighting energy, this measure degenerates to the aged sighting density.

Intuitively, energy density describes the temporal density of location information about a particular located-object. Thus, it incorporates aspects of both quality and availability. If sightings are infrequent, the energy density will be low. If sightings are coarse-grained or unreliable, the energy density will also be low. However, a bounded sighting density must be assumed.

6.2.4 Movement trace properties

The quantitative measures proposed above need to be abstracted into qualitative properties in order to help make decisions and choices. For example, we envisage selecting suitable movement prediction heuristics based on quality of location trace.

Firstly, in order to choose a prediction heuristic, it is important to know whether the location information for a located-object allows a complete reconstruction of the movement path in the sampling space. This is reflected by the *continuity* property. Secondly, physical located-objects are subject to *inertia*, i.e. they follow some regular path until redirected by some kind of event. Whether this regularity caused by physical inertia is reflected in the sampling space or not is indicated by the *directionality* property.

In the subsequent discussion, we use the concept of a *Movement Plane*, which has been described briefly in § 6.1.2. Here it is only necessary to know that a movement plane consists of a set of disjoint locations, along with a local definition of what constitutes a straight-line (specified by the function $Dir^{\mathbf{M}}$). An event trace is restricted to a movement plane by removing events that are irrelevant or too coarse-grained, and by relaxing the spatial resolution of events that are too fine-grained.

Continuity

An event trace **T** over a movement plane **M** is continuous if, and only if, there is at least one sighting in **T** for each visit of the located-object to any of **M**'s locations. A necessary precondition is that temporally adjacent sightings of **T** have spatially adjacent locations in **M** as defined by $Adj^{\mathbf{M}}$. By definition, located-objects move only between adjacent locations. Hence, a violation of the continuity property is indicative of an inadequate temporal sampling resolution.

Applying Nyquist's sampling theorem, a continuous movement trace can be constructed if the minimal sighting density $min(\delta_{\mathbf{n}})$ is greater than the maximum rate at which the located-object changes its location (assuming that locations are disjoint).

In an Active Badge system a continuous movement trace can be constructed if people always stay longer than 9 seconds in the area of a sensor, and if badge beacons are collected every 9 seconds.

The rate of location change depends on three underlying factors: the speed of the located-object, the size and shape of the locations, and the coherence of the movement (straight-line vs. oscillation). If the movement patterns of located-objects are extremely constrained (e.g. cars on a motorway), the rate of location change can be determined analytically. Otherwise, we must resort to statistical observation or simulation methods. Once the expected rate of location change has been determined, an adequate temporal sampling resolution can be chosen to guarantee trace continuity.

After a trace has been acquired, we believe that in most cases non-continuous traces can be identified simply by using the adjacency graph $Adj^{\mathbf{M}}$ to detect spatial gaps in the location trace. This should allow an informed choice of whether to use heuristics that require a continuous trace.

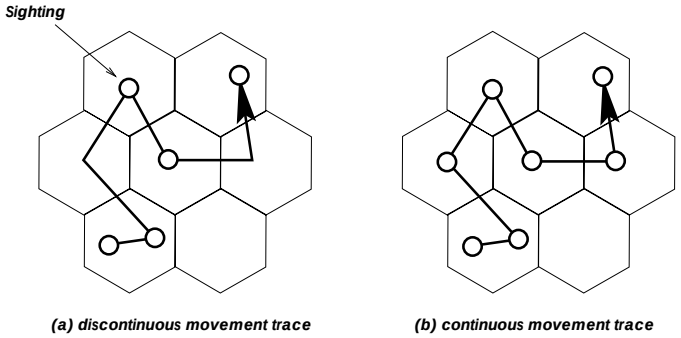


Figure 6.3: Movement trace continuity

Figure 6.3 illustrates the continuity property: the left-hand trace is not continuous because the located-object appears to "jump" between the second and third sighting.

Directionality

We call a movement trace directional with respect to a movement plane **M** if, and only if, each movement segment of the located-object touches at least three different locations of **M**. That is, in a directional movement trace each segment has a start location, an end location, and at least one intermediate location.

A movement segment is a period of time where the located-object travels with a non-zero speed along a straight line as defined by $Dir^{\mathbf{M}}$. It may correspond to a straight-line movement of the located-object in reality, but **M** could also define other directed movement patterns as segments. For example, a movement plane could be constructed that allows the "journey to work" to be a single movement segment.

While continuity of location information can always be achieved by choosing a low-resolution movement plane, directionality imposes a lower bound on both the spatial and the temporal resolution (for a given set of movement patterns).

A movement trace is directional with respect to a movement plane **M** if the trace is continuous and if the minimum length of a movement segment is more than twice the maximum diameter of the locations in **M**. This is a sufficient pre-condition for directionality.

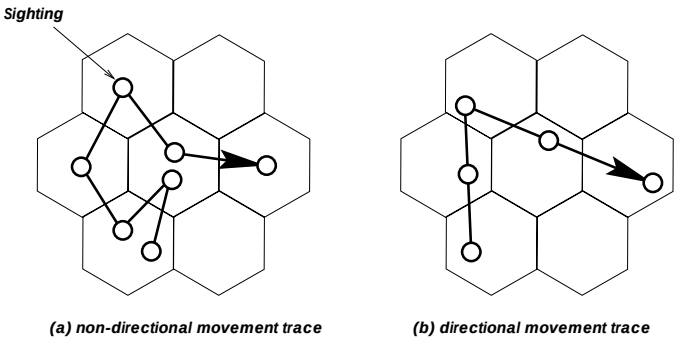


Figure 6.4: Movement trace directionality

Figure 6.4 illustrates the concept of directionality. The left trace is not directional because the spatial resolution is too coarse for intermediate locations to exist.

In general, we cannot distinguish a directional trace from a non-directional trace by observation in the sample space. However, oscillations between two locations (Figure 6.4) would indicate a non-directional trace.

Discussion

Continuity and directionality are two higher-level properties of location traces with respect to a given Movement Plane. The continuity property indicates that the located-object's movement can be modelled faithfully (i.e. without missing intermediate locations) as a

state machine. The directionality property indicates the state machine is non-Markovian: the next state depends not only on the current state, but also on the previous state(s).

6.3 Prediction

Predicting future locations requires assumptions about the mobility of a located-object. Such assumptions represent knowledge of constraints that in reality limit the mobility of located-objects. Common assumptions include:

- A located-object can only be in *one place at a time*. However trivial this may seem, this is a fundamental property of a located-object.
- *Located-objects do not move faster than a maximum speed.* While the speed of light is the ultimate limit, there are often more precise assumptions that can be made if the mode of transport is known. For example, cars on the road will not move faster than 250 kilometres per hour (except perhaps in Germany).
- *The movements of a particular located-object are repetitive to some degree.* Since located-objects mostly move in an environment with plenty of static stationary features (Rooms, Buildings, Streets, etc.), there are often certain paths which the located-object will traverse more than once.
- *The movements of a set of located-objects are repetitive to some degree.* Among a set of located-object with similar characteristics, some paths will typically be shared (for example the subway between underground station and college).
- *Located-objects' movements are influenced by scheduled activities and events.* Scheduled events are future events that are extremely likely to happen at a particular time, and each event affects the movements of the located-object in a specific way. For example, the closure of a bridge during a certain period would force a detour.

Each of these assumptions effectively defines a body of knowledge which can be applied in order to predict future movements of a located-object. The usefulness of each assumption and the associated heuristics depend on the target scenario. For example, when the sampling resolution is too low to identify movement patterns, prediction based on repetitive patterns will not be successful.

The available knowledge of a located-object's movements consists of the following components:

- The set of movements possible in a certain environment. This knowledge is represented by the movement plane, or more generally, by a set of movement planes.
- The history of its past movements. This knowledge is represented by a movement trace.
- Expectations about its behaviour. This knowledge is embodied by a set of heuristics.

Predictions are made by combining factual knowledge (movement plane and trace) with knowledge gained by induction or deduction (expectations).

Our approach is to use the movement plane as the reference framework for predictions. Movement traces and heuristics are always applied in the context of a specific movement plane to limit complexity and scope. It is also possible to employ multiple movement planes in parallel and combine their results.

Further, we utilise the quality of the movement trace with respect to a certain movement plane as a heuristic to choose the actual prediction heuristic. Thus we hope to target prediction heuristics at those scenarios they are most applicable to.

6.3.1 Prediction heuristics

We distinguish *inductive* heuristics, which incrementally predict the "next" location, and *deductive* heuristics, which do not rely on constructing a continuous trace between the present and the prediction.

Inductive heuristics

Central to an inductive heuristic is the notion of a current state (or location), which is used as a base to predict a "next" state. Once a "next" state can be predicted, longer-term predictions can be made by chaining basic induction steps.

Naturally, induction heuristics crucially depend on accurate information about the current state. Also, induction is prone to accumulation error. On the other hand, short-term inductive predictions can be very accurate [37].

Different inductive approaches are distinguished by their notion of state, and by the methods used to compute the "next" state. In the following we describe some of them.

Trajectory projection A located-object tends to possess inertia: its movement follows a certain path unless disturbed by an external event. The simplest variant of this heuristic is the calculation of a future position based on current location and velocity. The projection can be made for arbitrary points into the future with diminishing accuracy.

Typically, a trajectory projection requires knowledge of current speed and direction. Sometimes, location sensors will provide this information along with the position (e.g. GPS). Otherwise, those measures can be derived (with less confidence) from the movement plane $\mathbf{M}$ in combination with the movement trace. Here, $Adj^{\mathbf{M}}$ and $Dir^{\mathbf{M}}$ would respectively be used as a basis for the computation of speed and direction. Naturally, this is only feasible for high-quality (i.e. continuous and directional) movement traces.

In effect, a trajectory projection is a one-step induction from current to target state. State includes location, speed, and direction of the located-object within a movement plane.

Markov process In a Markov process [12], a discrete random variable changes continuously over time. The next state of this variable depends *only* on its current value. Prediction using a Markov process is a multi-step induction where the located-object's location is its state.

Location change can be treated as a Markov process if it is assumed that there is no observable inertia affecting the movements of the located-object. This may be justified, for example, if the sampling resolution is too low for inert movement segments to be visible.

Markov processes lend themselves both to simulation and to analytical solutions. Also, a Markov model can be constructed automatically from historical data. Given a movement plane $\mathbf{M}$ as a reference framework, a Markov model is represented quite easily by labelling the relation $Adj^{\mathbf{M}}$ with transitional probabilities.

As observed in [37], Markov processes alone are not suitable for all scenarios due to the fundamental assumption of ignoring the lessons of history! However, they have been used in combination with other heuristics to model random aspects of an object's movements. For example, Lam *et al.* [31] employ Markov models for one of several categories of mobility behaviour.

Pattern base If a located-object's movements can be represented as a sequence of state transitions, the sequence of visited locations lends itself to pattern matching. Thus, if the start of a pattern is recognised in the movement trace of a located object, the completion of this pattern can be used for location prediction. This is a multi-step induction where each state is characterised by a current location and a trace of past locations.

Liu [37] describes a pattern base consisting of *traces* and *circles* of subsequent states. The contents of the pattern base are matched against the movement trace using a number of metrics[3]. Also, patterns are extracted dynamically from historical data and added to the pattern base.

Patterns can vary in their generality. Sometimes, the pattern will be expressed as a sequence of concrete locations [37]. It is also conceivable to have more generic patterns (such as home-work-pub-home) that use generic rather than concrete locations. Also, grammars (such as regular expressions) could be used. Interestingly, it is also possible to interpret a trajectory as a movement pattern.

In contrast to the Markov model, a pattern-base approach is most suitable if inert movement segments typically cross multiple locations (i.e. the trace is directional). This is because a pattern normally needs to be matched with more than one location.

Deductive heuristics

Sometimes a located-object's location can be predicted for a certain time in the future without constructing a path from current to predicted location. This avoids the error accumulation typically associated with inductive methods.

Also, deductive methods do not make strong assumptions about the quality of the available location information. Thus, they can often be applied when inductive methods fail.

Application level itineraries and goals Certain applications are concerned with scheduling and planning of future activities of located-objects (e.g. diaries, route-

[3]Equality of corresponding locations, trace duration, transition frequency (see [36], page 78).

planners). Other applications may have knowledge of events that will affect the movements of the located-object.

This application-level knowledge is potentially very useful for longer-term location prediction. The challenge is to allow the prediction system access to application knowledge. However, without standardisation, access will be feasible and worthwhile only to the most commonly-used applications (such as distributed diary systems). A further concern is that application knowledge may be sensitive, i.e. it must be protected from unauthorised disclosure.

Statistical profiles Given a location service, statistics on the movements of a particular located-objects can be obtained easily. For example, we can measure the frequency of visits to a particular location along with the average duration of those visits. In turn, statistics can be used to predict the time, likelihood and duration of a future visit.

Rose and Yates [56] suggest using time-varying mobile unit location probability distributions to predict the movements of located-objects.

Statistics offer a cheap and relatively powerful way to predict locations. The necessary information can be acquired automatically and transparently. Hence, the statistical profile is an approach that can be used widely.

Discussion

This section has described various methods of location prediction, which can be used individually or in combination. For example, rather than starting an induction from the current location, a deduced future location could be used. Conversely, deduction could help to verify or rank the results of an induction process.

6.3.2 Applicability

As indicated above, each heuristic has a scope within which it can be successfully applied. The scope of applicability is measured along the following dimensions: time frame, trace continuity, and trace directionality.

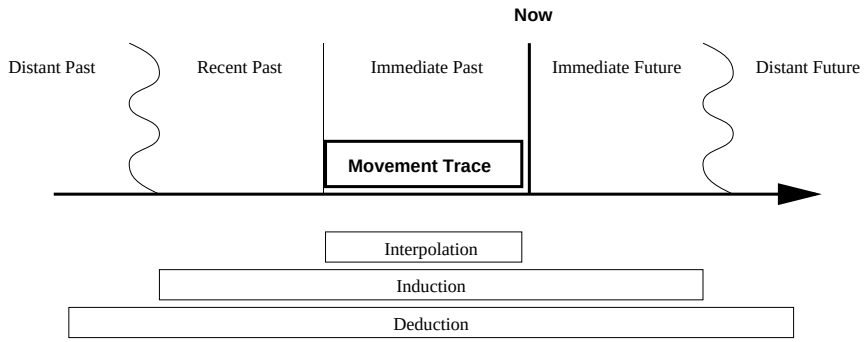


Figure 6.5: Time frames for location prediction

Time frame

For purposes of location-tracking, the current time signifies the end-point of the available location trace. With reference to this notion of current time, a request for location information falls into one of the following categories (figure 6.5):

- *Immediate future*. This time frame is perhaps most useful for location-aware applications. While no "hard" location data is available, the immediate future lends itself naturally to inductive heuristics. Deductive heuristics may be applied too, if an inductive approach fails.
- *Distant future*. The distant future requires prediction using deductive heuristics since error accumulation renders inductive methods ineffective.
- *Immediate past*. By definition, this time frame covers the available location trace. Hence only interpolation may be required.
- *Recent past*. This time frame precedes the immediate past time frame. Backward induction is possible.
- *Distant past*. No trace information is available, and no backward induction is feasible. Similarly to the distant past, deductive heuristics must be applied.

We conclude that for the recent past and the immediate future, induction is the most effective prediction method. Deduction can be applied, but it will be less effective due to the lack of reference to the current location. For the distant past and future, constructing a path from the present location is too error-prone, and only deduction is applicable.

Trace continuity

The continuity property of a trace over a given movement plane $\mathbf{M}$ indicates whether sufficient location information is available to faithfully track a located-object in $\mathbf{M}$.

Continuity is necessary in order to model the movement trace as a sequence of state transitions over $Adj^{\mathbf{M}}$. State machines are the basis for some induction heuristics (e.g. Markov process, patterns base). Therefore lack of continuity makes it difficult to apply those heuristics. Note that in some cases *interpolation* can fill temporal gaps and make a trace continuous.

Other heuristics should not be directly affected if the movement trace is discontinuous. However, lack of continuity may indicate generally poor availability of location information, and affects any aspect of a location service.

Trace directionality

The directionality property of a trace over a given movement plane $\mathbf{M}$ indicates whether the direction and speed of the located-object can be inferred from the trace. If continuity is lacking, interpolation may produce a trace that is both continuous and directional.

If a trace is directional, the final state of each movement segment depends on at least two preceding states. Therefore, if a trace is directional the Markov property does not hold.

Hence the movement trace cannot be treated as a Markov process. Equally, directionality indicates that the trace can be used for pattern matching and also for trajectory projections.

Directionality can be used to decide which inductive method to apply. On the other hand, directionality does not appear to have an influence on the applicability of deductive methods except that it indicates an adequate spatial and temporal sampling resolution.

Discussion

The time frame of prediction and the quality of movement trace have a significant impact on applicability ranking of prediction heuristics. Naturally, the more powerful heuristics (pattern base, trajectory projection) have the narrowest scope of applicability, while deductive methods offer lower-quality predictions in almost any scenario.

In a homogeneous system with known characteristics the best heuristic can often be chosen in advance for the system as a whole. Otherwise it is also possible to select dynamically the approach that is best suited for a given located-object or trace.

6.4 Moving from events to states

The previous section has outlined the importance of state-based mobility models for location prediction. Hence, there is a need to transform the sequence of location sightings gathered by the acquisition stack into a sequence of state transitions.

State transitions require an underlying state machine. We use the movement plane, or more precisely the adjacency relations Adj as the specification for this state machine.

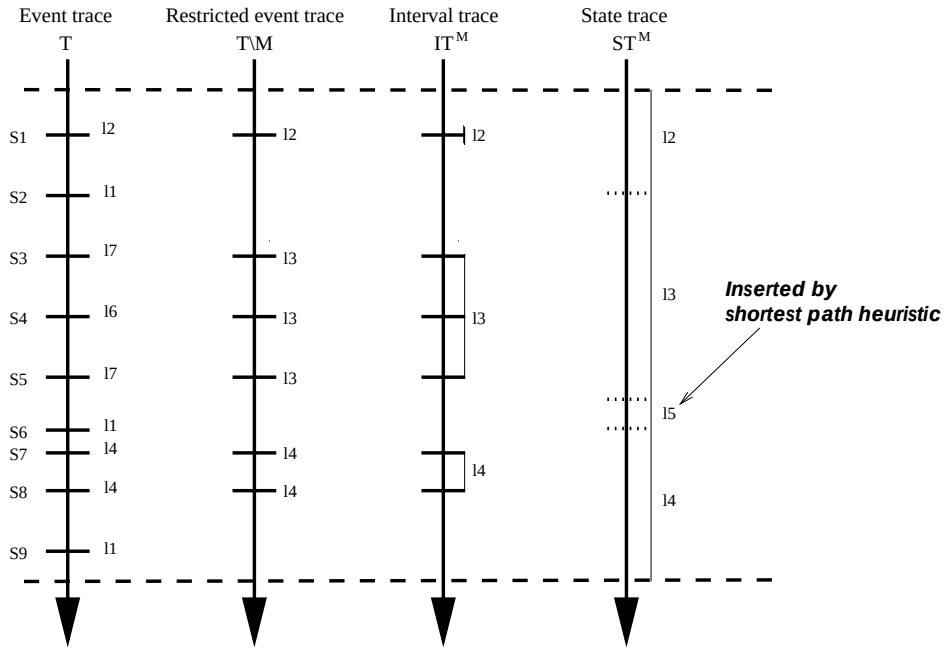


Figure 6.6: Translation of an event trace into a state trace

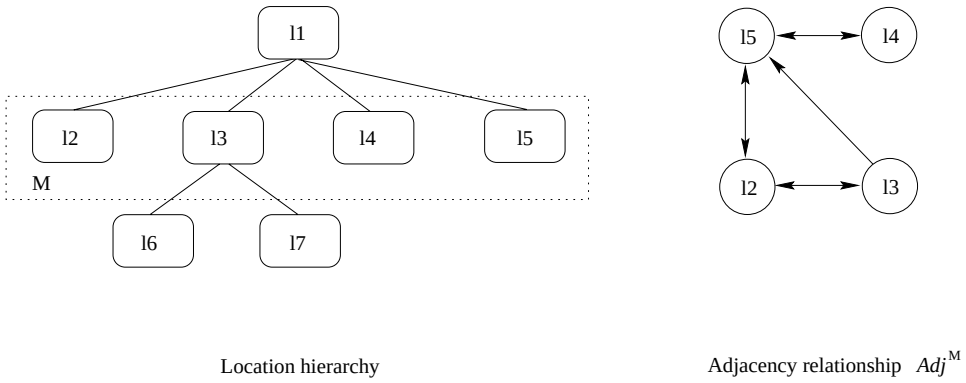


Figure 6.7: Location hierarchy and movement plane

Given the representations for interval traces **IT** and state traces **ST** defined earlier in this chapter, we identify the following stages of translating a trace **T** into a state trace $\mathbf{ST^M}$ over a movement plane **M**:

1. *Restriction* removes events from **T** that are irrelevant to **M**, it computes $\mathbf{T} \setminus \mathbf{M}$.
2. *Consolidation* translates the restricted event trace $\mathbf{T} \setminus \mathbf{M}$ into an equivalent interval trace $\mathbf{IT^M}$.
3. *Interpolation* produces a state trace $\mathbf{ST^M}$ over **M** by closing the gaps in $\mathbf{IT^M}$.

Figures 6.6 and 6.7 illustrate this process. The consolidation stage is facilitated since restriction limits the trace to a finite and mutually exclusive set of locations. Restriction has been described at the beginning of this chapter and shall not be discussed further. Consolidation and interpolation are described below.

6.4.1 Consolidation

The consolidation stage must translate a sequence of discrete sightings into a sequence of intervals. (An interval is the time span when a located-object is at a particular location.) Since consolidation is preceded by a restriction to a particular movement plane, we may assume that the locations occurring in the trace are mutually exclusive. Hence, an interval spans only events referring to the same location (rather than its sub-locations). Our consolidation algorithm works as follows:

1. Each event forms a primitive interval with duration zero.

$$\mathbf{I(s)} = (\mathbf{t^s}, \mathbf{t^s}, \mathbf{l^s})$$

2. Adjacent intervals are joined if they refer to the same location.

$$\mathbf{I_A} + \mathbf{I_B} = \begin{cases} (\mathbf{t_s^A}, \mathbf{t_e^B}, \mathbf{l^A}) & \text{when } \mathbf{t_s^A} < \mathbf{t_s^B} \\ (\mathbf{t_s^B}, \mathbf{t_e^A}, \mathbf{l^A}) & \text{when } \mathbf{t_s^B} < \mathbf{t_s^A} \end{cases}$$

3. Step 2 is repeated until no more joins are possible,

Figure 6.6 gives an example. The consolidation process results in an interval trace over $\mathbf{M}$. If the original trace was continuous over $\mathbf{M}$, and if the located-object stayed within the scope of $\mathbf{M}$ for the whole trace, the computed interval trace will have no gaps. Otherwise, remaining gaps can be closed by interpolation if possible. The resulting continuous interval trace is a state trace.

6.4.2 Interpolation

If consolidation does not produce a continuous trace, two explanations are possible:

1. The located-object went out of scope of the movement plane during the trace. Detection of this is outside the scope of this algorithm. We assume that the located-object remains in scope for the duration of a trace.

2. The temporal sampling resolution was too coarse to record a continuous trace over $\mathbf{M}$. In this case, it may be possible to close some or all of those gaps using heuristic interpolation.

Our approach assumes that the adjacency relation $Adj^{\mathbf{M}}$ accurately models all the potential transitions among locations. Thus, the following interpolation algorithm can be executed:

1. Identify a gap as a pair $(\mathbf{I_s}, \mathbf{I_e})$ of neighbouring but non-bordering intervals.

2. For each gap, find the set $\mathbf{P}$ of all possible cyclic-free paths between $\mathbf{l}^{\mathbf{I_s}}$ and $\mathbf{l}^{\mathbf{I_e}}$. If $\mathbf{P}$ is empty, we have an error condition. If $\mathbf{P}$ is large, interpolation will only have a small chance of yielding a correct result.

3. Apply a selection metric to each member of $\mathbf{P}$ in order to choose the most probable path between $\mathbf{l}^{\mathbf{I_s}}$ and $\mathbf{l}^{\mathbf{I_e}}$.

4. Insert the selected path into the trace. We have to choose the timing of the transitions according to some heuristic.

This algorithm can be performed using a number of path selection and path timing heuristics. If state transition probabilities are available (Markov model), they can be used to choose the most probable path of the located-object. If a pattern base is available, pattern-matching can be applied to generate interpolated paths. Otherwise, the selection of the shortest-path (either in terms of geometric distance or number of state transitions) appears to be a promising solution.

A pattern base can be also be used to determine the timing of interpolated state transitions. Otherwise, we have to settle for a notion of constant speed, either relative to geometric distance or the number of state transitions. Again, this should work reasonably well for short gaps.

If successful, interpolation produces a continuous trace, i.e. a state trace. However, it appears that interpolation only works well if there are very few probable alternative paths

to choose from. How good this interpolation is depends on the quality of the original trace. It is fairly easy to smooth over minor deficiencies in a basically continuous trace. If the located-object's movements are very constrained (e.g. a car on a motorway), it is even possible to compensate for longer gaps. In most other cases, interpolation will have so many alternatives to choose from that it becomes a lottery. Therefore we suggest using the trace metrics described earlier in order to decide whether interpolation is appropriate.

6.4.3 Summary

If a movement trace is of high-quality with respect to a particular movement plane, the sighting trace may be translated into a continuous sequence of state transitions. This translation involves restriction, consolidation, and interpolation. The motivation for translation is that many mobility-models are based on state machines.

6.5 Chapter Summary

Location information is only available for the past, which is not immediately useful for most location-aware applications. Hence, prediction and interpolation are employed to make constraints on the availability of location data transparent to applications. Thus, uncertainty is reduced by injecting "external" knowledge in the form of prediction heuristics.

Information about the position of a located-object accrues at discrete points in time. Hence, we have proposed the notion of a time-ordered event trace as a general model for the location data gathered by the acquisition stack. The quality and availability of location data represented by such a trace can be quantified using a number of measures. While the applicability of these measures depends on the target scenario, we have identified the notion of a sighting's *energy* as an interesting model to rank sighting quality. Consequently, energy density and average energy can be used to measure availability and quality over a trace of sightings. We have applied the measure of aged sighting density (a special case of aged energy density) to continuously measure the availability of location information for Active Badges.

Quantitative measures are useful tools but require some idea of the expected outcome. Our discussion has shown that for large heterogeneous systems such expectations cannot be identified for the system as a whole but only for parts of it. To alleviate the resulting complexity, we have proposed the more abstract properties of *continuity* and *directionality*, defined relative to a directed graph of mutually exclusive locations as represented by the *movement plane* model.

In order to predict location, several inductive and deductive location prediction heuristics are available. Inductive heuristics, such as Markov models or a pattern base, are useful for accurate short-term location prediction. However, they are state-based and thus require a continuous and (in the case of pattern-based approaches) directional trace. Deductive heuristics do not rely on the trace itself but solely on "external" knowledge such as long-term statistics or scheduled events. Hence deduction is always available but single-case accuracy may not be very good.

Due to the relative importance of state-based heuristics, we have described a way of translating an event trace into a sequence of state transitions over a movement plane. During this process, it is often necessary to interpolate between location events. Such interpolation can utilise state-based prediction heuristics, such as Markov chains or a pattern base, in modified form.

This chapter has not championed any particular method of location prediction. Our aim has been to outline a framework and a set of tools to build a prediction system to suit a particular target environment. The framework is defined by the hierarchical location model, the movement plane model, along with the representation scheme of traces for events, intervals, and states. The tools are the various uncertainty metrics over events and traces, plus the heuristics for prediction and interpolation.

Chapter 7

Security Considerations

Security of information systems is of great concern to individuals and organisations for differing reasons: commercial organisations are mostly interested in the integrity of their data, the military worries more about secrecy (see [9] for a discussion), and individuals are concerned about *privacy*, broadly personal control over the secrecy of private information.

In this chapter, we consider the security requirements of a location service. The spectrum of application for a location service is wide, ranging from mobile telecommunication systems to emergency assistance services and computer-supported cooperative work. As a result, location services will often become repositories of potentially sensitive personal and corporate information. *Where you are* and *who you are with* are closely correlated with *what you are doing*. To leave this information unprotected for everybody to see is clearly undesirable. People would feel uncomfortable if their every move could be watched anonymously. Similarly, businesses would probably not like the idea of competitors or staff monitoring the attendance of every meeting. Further, location data will be used, directly or indirectly, as input for decision-making processes. Hence, the integrity of location data is also important.

We conclude that location information needs to be protected against unauthorised disclosure and modification. However, the exact level of protection varies widely from context to context. Personal location services, corporate location services, and military location services will all have different requirements for secrecy and integrity. Hence, we concentrate in this chapter on models for specifying security. Those models can then be used to address the requirements of a specific location service.

We shall focus on the secrecy aspects of security. We outline two typical deployment scenarios for a location service. Then, we explore the application of traditional mandatory and discretionary security mechanism to our problem. We conclude with a brief description of our prototype implementation, along with a discussion of related work.

7.1 Requirements

We outline two usage scenarios for a location service, which highlight different deployment environments and the resulting sets of requirements. We are especially concerned with the balance between security imposed by the system (mandatory security), and security

specified by individuals (discretionary security).

7.1.1 Scenario I: Organisational location service

Within organisations, there is often the need to locate people in real-time. For instance, trucking companies often use GPS-based systems to reroute vehicles efficiently. Further examples include computer-supported collaborative work, communication with mobile workers, and location-based security mechanisms. Their common theme is that acquisition, management, and use of location information are ultimately controlled by a single decision-making body, or organisation.

Security policy is set by the organisation for the whole location service. Typically, local discretion is permitted only within the bounds defined by organisational policy. The system is closed to outside access (except for limited and controlled cases). The people and mobile objects to be tracked are registered with the organisation. The coverage area, however, may well be very large if tracking technology and communication networks permit.

In this setting, both integrity and accuracy of location information are of importance, since the organisation's processes and decisions will be affected. It should be possible to tell whether information is at least trustworthy. Some applications demand a high degree of trustworthiness from location information, while others may well trade availability of information for accuracy.

As far as secrecy within organisations is concerned, we see two basic requirements: location-centric and user-centric privacy[1]. Firstly, an organisation may want to allow only a certain group of people to discover who is in a specific room or building. For example, a floor of a building might be 'open' to everyone who works there, but not to people from other floors and buildings. Secondly, a person's location should probably only be visible to a restricted group. For example, the managing director might be visible all the time to his or her secretary, while other people can see him only when he is in his office.

At present, most location services fall into the organisational category. However, provision of a global, public location service requires a more general, inter-organisational approach.

7.1.2 Scenario II: Global location service

In contrast to the well-controlled, relatively closed environment described above, we now discuss the scenario of a *global* location service. We expect such a service to be provided by a network of cooperating providers, similar to today's mobile telephone system. The providers would have roaming agreements with each other. Subscription would be necessary in order to be tracked by the service, and also to access the service. Service level and security provisions would be governed by legal contract.

The applications for such a service are the same as described in the previous scenario. Further, there is scope for third-party location-aware services. For example, such a service

[1] User-centric privacy is not the same as personal privacy, but rather user-centric organisational privacy. Personal privacy is an orthogonal concept.

might be responsible for automatically informing emergency services when a distress signal from a subscriber is received. Users will have to trust the service providers to obey the security policy laid down in the service contract.

The global location system is open in two ways. Firstly, roaming subscribers may encounter service providers that they have not met before. Secondly, service providers may encounter unknown subscribers in their area. There could also be competing service providers in the same area, further complicating matters. The problem here is mainly one of cross-domain authentication and user profile management, which is outside the scope of this thesis.

We envisage that service providers will be obliged to implement certain generic security policies, such as nondisclosure to unauthorised third parties. Additionally, subscribers would specify an acceptable security policy for themselves. For example, someone might choose to be visible to their boss at work but not at home or at weekends. These policies would presumably be mostly user-centric, while location-centric policies (for example, to protect the privacy of a person's home) could be useful, too.

Ideally, there should also be more generic ways to specify access authorisation. For example, when attending a conference I would like to be visible to all the other attendees without actually knowing them. Similarly, I might wish to be anonymous in locations matching a given constraint, such as a motorway.

Generic authorisation constraints are especially important since the service is partitioned among many providers. These providers must rely on local knowledge to make access control decisions. Constraints that require frequent access to non-local information cannot be considered a scalable solution to this problem.

The requirements governing integrity and accuracy of the location information can also be expected to vary widely. Even a single subscriber could have multiple accuracy requirements for different applications.

7.1.3 Discussion

In both scenarios, secrecy is the main concern. In the first, secrets of the organisation need to be protected while individuals' privacy is of lesser concern. In the second, subscribers' personal privacy is the main requirement. In each case, privacy has user-centric and location-centric components.

A location service also needs to be protected from false location data. Further, there is a need in both scenarios to distinguish trusted from untrusted location information. The integrity of trusted information needs to be protected against improper modifications.

In the remainder of this chapter, we focus on models for specification of secrecy constraints applicable to both scenarios.

7.2 Access control for a location service

Functionality and manageability of a location service depend primarily on the location model used. We argue that manageability (and security) are facilitated by symbolic location abstractions.

A location service over a symbolic location model offers the following basic functionality:

- *Given a located-object, return all the current symbolic locations of this object.*
- *Given a symbolic location, return all the located-objects currently located there.*

The difficulty in specifying a security model over these two functions is indicated by their symmetry. Either function can reveal all available location information. Hence access control for both functions must be consistent.

As far as the architecture of the location service is concerned, many different processing and distribution models are possible. However, in this chapter we shall focus on architecture independent security models.

In the following, we discuss how security for a symbolic location service with those two functions can be specified. We expect that the results can be applied to location models including geometric data and to location services with more complex functionality.

7.2.1 New challenges

Location information essentially consists of fast-moving dynamic relationships between multiple objects. Difficulties for existing approaches are the dynamism of the information, and the fact that location information does not consist of knowledge of objects, but knowledge of object relationships.

The first difficulty arises because management systems tend to rely on a relatively static framework, e.g. the domain-based management framework described in [72]. Here, the problem domain is structured into a graph of management domains, each domain containing a set of references to managed objects. Managers are expected to explicitly add objects or remove objects from a domain. We believe that while the domain graph should remain mostly static, dynamic location-dependent domain membership is required to manage mobile objects. However, this is more an architectural problem and lies outside the scope of this work.

The second difficulty mentioned above is the actual motivation for writing this chapter. We seem to be unable to specify security policies for location information using the standard models for access control, Lampson's access matrix [32] and Bell-LaPadula's security labels (see [7]).

Traditionally, the use of access control is either mandatory (imposed by the system), or discretionary (left to the owners of the objects). Both approaches are based on the subject-target paradigm. With mandatory access control, a subject is allowed read-access or write-access to a target if certain axioms over the security labels of subject and target are satisfied. When using discretionary access control, an access matrix [32] with possible subject-action-target combinations is constructed. Access by a subject to a target with an action is granted if the corresponding combination is a member of the access matrix.

With location information, there is no obvious target object. If the located-object is treated as the target object, it is hard to specify access control for all objects at a given location. If the location is the target object, it is difficult to specify access control for

a given located-object. Using both methods in combination is not satisfactory because access control information is duplicated.

In the remainder of this section, we describe how the classic access control models can be generalised to cope with this problem. For the purpose of the discussion, we shall assume a domain-based framework with dynamic location-dependent domain membership.

7.2.2 Matrix-based access control

In domain-based frameworks, matrix access control policies are specified by rules of the form:

`<subject scope>` { `<list of actions>` } `<target scope>`

Semantically, such a policy allows any subject from `<subject scope>` to perform one of `<list of actions>` on a target from `<target scope>`. This corresponds to an access matrix where both subjects and targets are domains. This additional level of indirection allows policies to be specified for groups rather than individual objects.

Application

As far as location information is concerned, a typical (informal) policy is

`Joe may see that Fred is located at Building@/School`

A policy with the same meaning is:

`Joe may see that Building@/School encloses Fred`

Clearly, both policies specify the same thing: `Joe` is allowed to observe a particular relationship, collocation, between `Fred` and `Building@/School`. However, such a policy cannot be expressed adequately in conventional subject-action-target paradigm. This limitation is somewhat alleviated by policies with additional constraints [44], allowing expression of the policy in canonical form:

`Joe { testForCollocation(PERSON) } Building@/School WHEN PERSON=Fred`

This specifies that `Joe` is allowed to perform the action `testForCollocation(PERSON)` on `Building@/School` when `PERSON` equals `Fred`. The action contains `Fred` as an implicit target. Unfortunately, this necessitates the evaluation of the `WHEN` clause at run-time. The `WHEN` clause contains an arbitrarily complex first-order logic expression, which makes a lightweight implementation somewhat difficult. Even worse, conceptual clarity is lost. The essence of the actual policy is obscured: granting authorisation for an action that, symmetrically, affects the rights of multiple targets.

To deal with this problem, we propose the use of multi-target policies of the form:

`<subject>` { `<action>` } `<target 1> ... <target n>`

The (informal) semantics of such a policy are: subject is authorised to perform action over the composite entity consisting of targets `1` to `n`. Obviously, multi-target policies are only useful for actions that affect multiple targets at the same time, such as binding of component interfaces in a distributed system by a third party [10], or arranging business deals ("match-making"). Applied to our example, this reads:

```
Joe { testForCollocation } Fred, Building@/School
```

For completeness' sake, multiple subjects can also be introduced:

```
<subject 1> ... <subject m> {  <action>  } <target 1> ... <target n>
```

Multi-subject authorisation policies describe authorisations for actions which require multiple subjects to perform an action together, such as opening a deposit locker, or authorising a cheque. A set of policies over `m` subjects and `n` targets corresponds to an `m+n` dimensional access matrix. An example:

```
Sweden, Finland { mediate } Israel, Syria
```

This policy specifies that `Sweden` and `Finland` may (together) mediate between `Israel` and `Syria`. This does not convey authorisation for either Sweden or Finland to mediate alone.

Such policies are necessary for actions that operate over dynamic relationships between a fixed number of objects. The objects in those relationships can collectively act as subject or target of an action. Arguably, the relationships themselves could be promoted to first class objects, leading to a more general solution. However, we believe the additional complexity is not justified here.

Note: The approach of multiple source scopes and target scopes is distinct from the additional grantee scope proposed in [83]. This scope is used to specify objects to which a policy can be delegated. Grantee-scopes are an extension facilitating the management of policies rather than extending their expressive powers. The approach described here is orthogonal and could be combined with grantee scopes.

7.2.3 Label-based access control

Mandatory access control in the domain framework is implemented by assigning security labels to management domains. All objects within a domain inherit the domain's label. If an object is a member of multiple domains, it inherits the least upper bound of all its parents' labels. Access is granted whenever the security labels of subject and target satisfy a certain set of axioms.

Analogous to the matrix-based case, the pair of single subject and single target does not in itself contain enough information to decide whether access to location information should be allowed. Hence, the security labels for both targets, that is location and located-object, should be consulted along with the label of the subject. Therefore, the axioms must cater for multiple subjects and multiple targets.

A label consists of a fixed number of attributes. There is an equivalence relationship defined over the set of values for each attribute. Further, attribute values may be partially

or totally ordered. These relationships are used by the axioms to establish a "dominates" relationship between labels. This in turn is also a partial ordering relation.

The most common label format, as used by Bell-LaPadula (see [7]), has two attributes. The first attribute, **S**, a sensitivity level (e.g. non-classified, confidential, secret, etc.), is totally ordered. The second attribute, **C**, is a set of categories. Examples of categories are nuclear physics, data encryption, and global positioning. Category sets are partially ordered by sub-set inclusion. A label **A** is said to **dominate** label **B** if both of **A**'s attributes are greater or equal to **B**'s corresponding attributes:

$$\mathit{dominates}(\mathbf{A}, \mathbf{B}) \Leftrightarrow (\mathbf{S^B} \leq \mathbf{S^A}) \wedge (\mathbf{C^B} \subseteq \mathbf{C^A})$$

For brevity, we shall use the object's name to refer to the object's label in the *dominates* predicate. Commonly used axioms are:

1. Subject **S** may `read` target **T** only if $\mathit{dominates}(\mathbf{S}, \mathbf{T})$
2. Subject **S** may `append` to target **T** only if $\mathit{dominates}(\mathbf{T}, \mathbf{S})$
3. Subject **S** may `overwrite` target **T** only if $\mathit{dominates}(\mathbf{S}, \mathbf{T})$ and $\mathit{dominates}(\mathbf{T}, \mathbf{S})$

These axioms ensure that information may only flow from objects with lower security classification to objects with higher classification. Thus, classified information cannot be declassified by 'normal' operations.

Application

We need to define a set of axioms over subject and target that allow a decision to be made whether access should be granted. In contrast to the approach described above, we need to deal with two targets: locations and located-objects.

We wish to express the following high-level policy for mandatory access control:

`Location data may be disclosed only if the secrecy of neither located-object nor location is infringed.`

In this context, "infringement of secrecy" translates to a flow of classified information to a target with lesser classification.

We attach security labels S, L, and O to subject, location, and located-object, respectively. The above policy can then be expressed as follows:

`S may see L at O only if` $\mathit{dominates}(\mathbf{S}, \mathbf{L})$ `and` $\mathit{dominates}(\mathbf{S}, \mathbf{O})$

The *dominates* relationship is a partial order because attribute values are drawn from partially-ordered sets. Therefore, instead of verifying the *dominates* relationship for each label, we may choose to combine all target labels into a single label. The combined label's level is intended to be greater than or equal to any of the individual labels. This notion corresponds to the mathematical concept of a least upper bound (*lub*) over a set S partially ordered by the *dominates* relationship. We define the *lub* over a set of labels as follows:

$$\begin{aligned} \mathit{ub}(\mathbf{S}, \mathbf{x}) &\Leftrightarrow (\forall\, \mathbf{y} \in \mathbf{S}).(\mathit{dominates}(\mathbf{x}, \mathbf{y})) \\ (\mathit{lub}(\mathbf{S}) = \mathbf{x}) &\Leftrightarrow \mathit{ub}(\mathbf{S}, \mathbf{x}) \wedge \neg\, (\exists\, \mathbf{z}).(\mathbf{ub}(\mathbf{S}, \mathbf{z}) \wedge \mathit{dominates}(\mathbf{x}, \mathbf{z}) \wedge \neg\, (\mathbf{x} = \mathbf{z})) \end{aligned}$$

In this definition $ub(\mathbf{S}, \mathbf{x})$ is a predicate that is true if $\mathbf{x}$ is an *upper bound* of the set $\mathbf{S}$ ordered by the *dominates* relationship. An upper bound $\mathbf{x}$ is not required to be a member of $\mathbf{S}$.

Using the *lub* of the participants' security labels, we can express the above policy as:

$$\texttt{S may see L at O only if } \mathit{dominates}(\mathbf{S}, \mathit{lub}(\{\mathbf{L}, \mathbf{O}\}))$$

The computation of the *lub* can be simplified because the ordering of security labels is based on the ordering of labels' attributes: sensitivity level and category set. The *lub* of two composite labels can be constructed from the *lub*s of the corresponding attributes of the label. That is, we compute the *lub* of the sensitivity levels and the *lub* of the category set. The label consisting of both results is the *lub* of the two original labels. More generally, the *lub* of a set of attribute tuples can be computed as the tuple of the *lub*s of the individual attribute values:

$$\mathit{lub}(\{(\mathbf{a}_1, ..., \mathbf{a_n}), (\mathbf{b}_1, ..., \mathbf{b_n})\}) = (\mathit{lub}_1(\{\mathbf{a}_1, \mathbf{b}_1\}), ..., \mathit{lub}_\mathbf{n}(\{\mathbf{a_n}, \mathbf{b_n}\}))$$

Note that the *lub* for each set of attribute values operates over the partially-ordered set specific to that attribute.

Axioms over operations with multiple subjects can be defined analogously. Here, the compound label of all subjects should be less than or equal to the individual labels. Hence, the compound label is defined as the greatest lower bound (*glb*) of the subject labels.

Consider the following policy with $\mathbf{T}$ and $\mathbf{S}$ defined as sets of objects:

$$\text{S may } \texttt{read} \text{ T if } \mathit{dominates}(\mathit{glb}(\mathbf{S}), \mathit{lub}(\mathbf{T}))$$

This policy permits a flow of information from a group $\mathbf{S}$ of objects to a group $\mathbf{T}$ of objects, provided that the least classified element of $\mathbf{S}$ still dominates the highest classified member of $\mathbf{T}$. This shows how the Bell-LaPadula security model can be applied to actions with multiple subjects or multiple targets. It does not really matter how many of the members of $\mathbf{S}$ or $\mathbf{T}$ are involved in a particular action. Hence, the proposed mechanism is applicable also to arbitrary groupings of objects, such as management domains.

When applying the proposed mechanism to the secrecy of location information, we use the pair of (location, located-object) as the set of target objects. A test for collocation is allowed if the source object *dominates* both target objects:

$$\texttt{S may testForCollocation(L,O) if, and only if, } \mathit{dominates}(\mathbf{S}, \mathit{lub}(\{\mathbf{L}, \mathbf{O}\})$$

This basically is a `read` operation. Since there is only one *dominates* ordering, it is not easily possible to specify more selective security policies, e.g. for the protection of personal anonymity. On the other hand, update authorisation can be specified:

$$\texttt{S may submitLocationUpdate(L,O) if, and only if, } \mathit{dominates}(\mathbf{S}, \mathit{glb}(\{\mathbf{L}, \mathbf{O}\})$$

This is an `append` operation, i.e. previous updates are not discarded. Updates that overwrite previous values are tricky because the corresponding axiom requires symmetric domination between source and target. In turn, this requires that both location and located-object have the same security label. This is impractical.

Effective protection by a mandatory security scheme requires careful design of the security labels, and a consistent labelling policy. Further work is required in order to establish the relevant design principles. However, we have shown that in principle, a generalised form of the Bell-LaPadula can be applied to the problem of specifying security for location information.

7.3 Mandatory vs. discretionary access control

From a functional point of view, mandatory label-based access control provides a simple framework for consistent protection of secrecy. However, individuals cannot selectively allow or restrict access to private information. Hence, label-based access control is not a suitable mechanism to protect people's privacy. It will therefore be favoured by organisations with strong internal security requirements. This simplicity, however, comes at the cost of reduced flexibility. Most environments, organisational or global, will require some kind of matrix-based access control. These controls can be specified at the organisational level, or by the owner of the information.

From an administrative point of view, mandatory labels require a central authority for creating and assigning security labels. Therefore, a mandatory scheme is not suitable for decentralised environments, such as described in scenario II. Matrix-based systems can be administered either centrally (scenario I) or in a decentralised fashion (scenario II). In general, central administration is less complex but not always practical.

We believe that few environments will rely solely on mandatory label-based access control to location information. The needs of most scenarios can be satisfied with matrix-based access controls, perhaps using management domains and a policy notation as supporting framework. Further, security labels can be emulated by policies. The converse does not hold.

7.4 What kinds of access control policies are needed?

No single monolithic access control scheme appears to work well in all (or even most) environments. Therefore, we advocate a mix-and-match approach, which allows orthogonal fine-grained access control mechanisms to be chosen and combined according to the actual security requirements. Additionally, conventional access control mechanisms can be applied to protect the location service as a whole.

Our protection mechanism is structured into three layers: control of access, control of visibility and control of anonymity. In an actual system, only one or two of these layers might be used. Semantically, the authorisation granted by one layer is only a necessary precondition for the actual access authorisation. It can be overridden by higher layers. In the following paragraphs, we describe the functionality of each of the layers.

Access policies

Access policies specify the traditional level of access authorisation. That is, unauthorised queries are rejected. However, in order to achieve fine-grained access control query results

must also be considered. A single query can produce a perfectly-authorised result in one set of circumstances, and an unauthorised result in a different set of circumstances. Therefore, results that are unauthorised need to be removed from the result set. Only queries that cannot possibly produce authorised results should be rejected straight away. The decision whether a query should be rejected must not allow anything to be inferred about the affected locations and located-objects. Therefore, this decision should be made without reference to objects' current locations.

```
Joe { accessCollocation } Fred, Building@/Hux
```

This policy states that `Joe` is allowed to observe collocations between `Fred` and `Building@/Hux` (and all of its sub-locations).

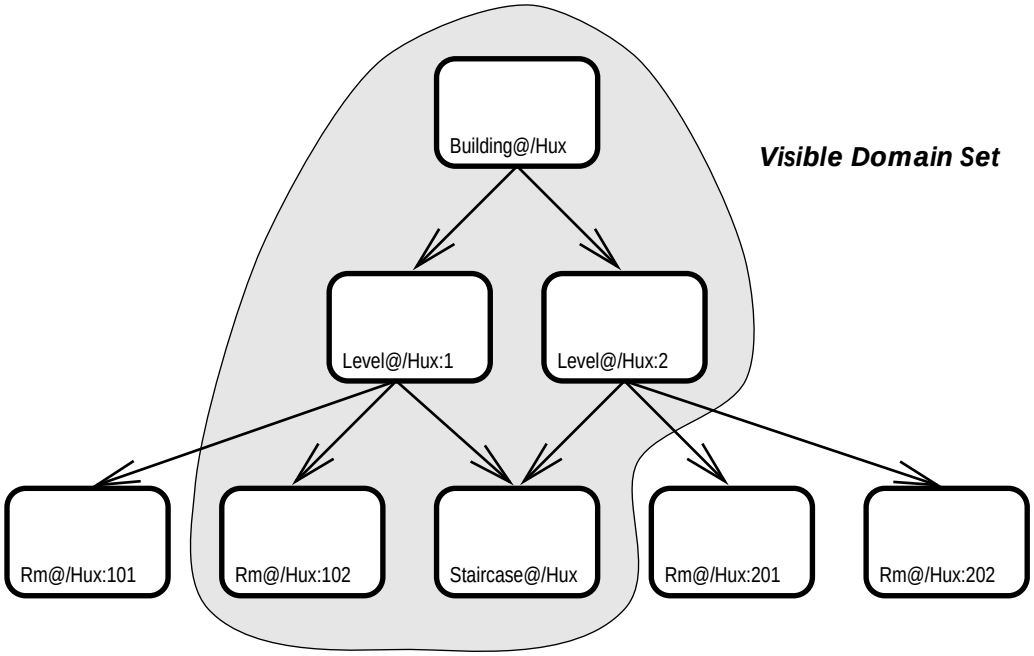


Figure 7.1: Location hierarchy with a visible domain set

Access policies specify necessary pre-conditions which may be strengthened by policies governing anonymity and visibility. We think of access policies as the "ring fence" surrounding the "playground" of the visibility policies and anonymity policies described below.

Visibility policies

Visibility policies control the level of location detail released. These policies typically act as filters replacing detailed location information with less detailed information. Such a substitution is made possible by the hierarchic structure of the location domain space as shown in Figure 7.1. For example, if a query yields the unauthorised result `Level@/Hux:201`, the result is replaced by `Level@/Hux:2`. The set of visibility policies for a given subject and located object defines the set of visible location domains.

```
Joe { accessLocation } Fred, Level@/Hux:1
```

This policy states that a co-location between `Fred` and `Level@/Hux:1` (including its sub-locations) may be observed as `Level@/Hux:1`. The policy does not specify whether access is allowed or whether the identity `Fred` should be revealed.

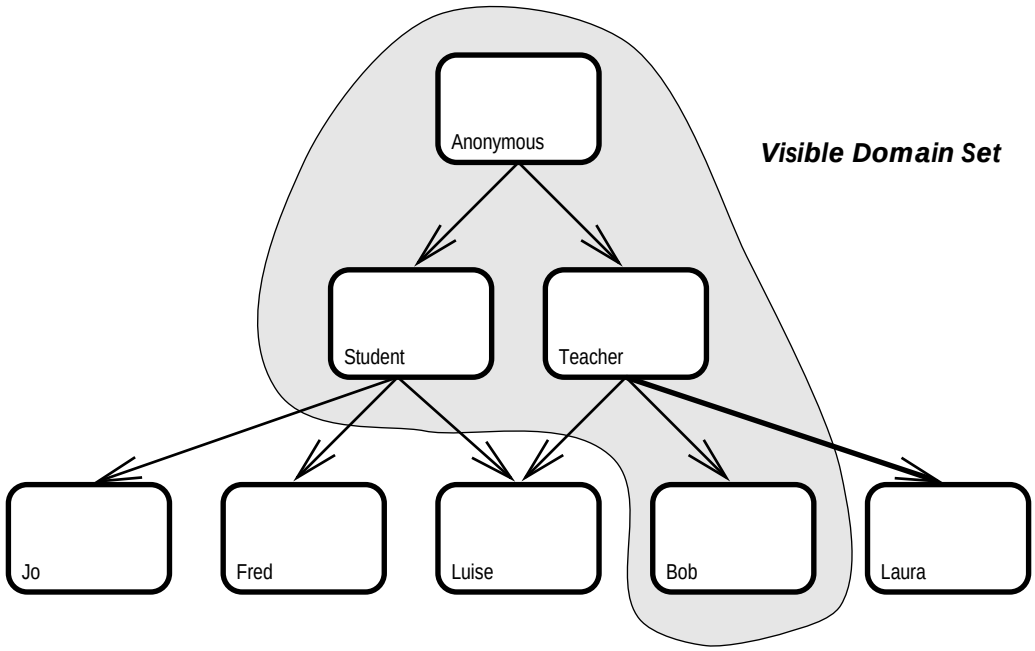


Figure 7.2: Located-object hierarchy with a visible domain set

Anonymity policies

Anonymity policies control the level of detail released about the identity of an object at a particular location. This is conceptually similar to the visibility policies described above. Instead of using a hierarchy of locations, we employ a hierarchy of identities as shown by Figure 7.2. The ordering of identities for a given located-object reflects increasing anonymity. Therefore, we can automatically replace concrete identities with more anonymous identities in a result set without affecting its correctness. A set of anonymity policies for a given subject and location defines a set of visible identities.

```
Joe { accessIdentity } Anonymous, Building@/Huxley
Joe { accessIdentity } Fred, Building@/Huxley
```

The first policy states that within Huxley Building (and its sub-locations), everybody should be visible as anonymous. The second policy specifies an additional but non-conflicting authorisation to `Joe` allowing him to see Fred as `Fred` in Huxley Building and all its sub-locations.

Higher-level policies

The three levels of access control can be combined in different ways to implement higher-level organisational or personal security policies. Examples include:

- *Correlate publicity of location with anonymity.* Such a policy corresponds quite closely to our intuition about privacy. In a public place, we expect to be anonymous, whereas everybody knows our identity when we are in our office. Such a style of access control can be specified directly using anonymity policies.

- *Correlate enquirer's role with the revealed granularity of location.* Actually, we would like to correlate the purpose of a query with the granularity of the result, but this is hard to do directly. Fortunately, the enquirer or his role are often a good approximation for the purpose of the query. This high-level policy can be refined using the visibility policies described above.

- *Do not allow outside access.* In type I scenarios, we expect this to be a common high-level policy. While this could be expressed using visibility policies or anonymity policies (or rather, their absence), we prefer not to over-complicate things. Access policies offer a simpler and thus more suitable mechanism to specify "hard" access control (as opposed to "soft" access control with anonymity and visibility policies).

These examples show that the proposed policy types offer significant flexibility, thus enabling them to address the requirements of a range of organisational contexts from both scenarios. Larger case-studies are required to evaluate the practical suitability of our approach.

7.5 Prototype implementation

We have implemented the protection model described above in a database-centric location service (cf. appendix D). Each of the protection layers — access, visibility, and anonymity — is enforced by a corresponding filter implemented as a stored procedure.

A filter is configured by access rules stored in a database table. It enforces access control by removing unauthorised elements from result sets. Filters are idempotent and can be chained.

- Access policies were implemented by computing the geographical area of authorised access for each pair of subject and located-object. At run-time, the filter checks whether the located-object is within the authorised area. This has the advantage of by-passing the hierarchical data representation but assumes that the location structure is static. If no area of authorised access is known, the query can be rejected right away.

- Visibility and anonymity policies are implemented as filters over result elements. For each element, the relevant tables are checked to see whether it is authorised. Since our result sets are complete (i.e. contain all higher-level domains), we can simply

remove unauthorised result elements. This is straightforward but perhaps not the most efficient solution.

This prototype has allowed us to verify the consistency of our protection model. Unfortunately, we have met performance problems caused by the evaluation of visibility and anonymity policies. Better performance could be achieved if the filters were working over a result set as a whole rather than over each element in turn.

7.6 Previous approaches to location service security

Typical commercial location service implementations, as used by GSM [48] for example, need to offer strong guarantees for the secrecy of location data. Secrecy is ensured by closing the system to outside access and by coarse-grained traditional access control (if there is any). More sophisticated approaches have been proposed by the research community.

Researchers at Xerox PARC where among the first to recognise the security implications of a location service [67]. In [68] they argue that different environments need different levels of protection for people's privacy. They also advocate user control over the disclosure of location information. The approach allows for protection of anonymity via 'secret groups'. They argue that in a large, heterogeneous system only the user-agent approach (as opposed to the location service approach) can deliver a meaningful protection of privacy. Xerox PARC's user-centric architecture is spelt out in [66]. Here, the user agent implements the access control decisions as specified by the corresponding user. Access control can also be delegated to a central Location Broker to increase efficiency. Locations are not treated as first-class objects in this model, that is, no explicit policy regarding access to a specific location can be specified.

Rizzo *et al.* describe their work on a secure location service for an office environment in [55]. Their location service is constructed of a tree of Locators, which are location tracking subsystems. Secrecy is protected by access control to those Locators. Capabilities are employed to allow select access to Locators. (These capabilities could, in principle, be used to specify the range of authorised results for Locator queries.) Organisational policies are expected to be hardwired into the Locators, while discretionary policies can be specified and altered by the individuals who 'own' the location information.

In both cases, the location services have been designed with a concrete implementation-specific security model in mind. There is no general architecture-independent specification of security policy which could be applied to a different location service architecture.

7.7 Chapter Summary

In this chapter, we have discussed the security requirements faced by location services deployed in different organisational environments. We have identified two likely deployment scenarios, large organisations and heterogeneous global services.

Both mandatory label-based protection and matrix-based protection can be applied to a location service. In both cases, the traditional approaches need to be generalised in

order to be suitable for the location service. This is because location information does not provide an obvious target object for policies and labels. If the located-object is treated as the target object, it becomes very hard to specify an access control for all objects at a given location. If the location is made the target object, it is difficult to specify access control for a given located-object. Using both methods in combination is not satisfactory because the access control information would be duplicated. Therefore, we have proposed multi-target policies for discretionary access control, and three-label axioms for mandatory policies.

Matrix-based access control offers a flexibility and expressiveness far superior to label-based access control. When using a domain-based framework, the access matrix can be specified as a set of canonical policies over groups of objects. Thus, the policy-based approach becomes scalable and manageable. Further, both centralised and decentralised system can use policies. Label-based access control caters for a much narrower set of requirements. Therefore it is only appropriate for use in systems with very specialised requirements.

We have designed and implemented a policy-oriented security model over a location service based on a hierarchy of symbolic locations. Our model allows for flexible protection of organisational and personal privacy. We have identified three levels of protection: access protection, location anonymity and personal anonymity. These protection levels can be provided by either mandatory or discretionary access controls.

Chapter 8

Conclusions

This thesis has discussed the provision of location-awareness in an open distributed computing environment. For this purpose, it has identified the functional and structural elements of global and general location tracking. These elements should be provided to applications through a logically centralised location service.

While there are many more issues to be discussed and problems to be solved, we have focussed on what we believe are core problems: data model, functionality, architecture, and security.

The next section summarises the contribution of this thesis, followed by a discussion of areas worthy of future investigation.

8.1 Recapitulation

8.1.1 Location-Awareness

Users and applications require information about the physical location of real-world objects. They require knowing their own location, or the location of other objects. Location-awareness comprises both.

Location-awareness should not imply location-sensor awareness. Hence, we propose a level of indirection between location sensors and location-aware application. If sensors and applications are not connected to the same computer, this indirection layer is encapsulated by the abstraction of a location service.

8.1.2 Location model

It was recognised early on that the chosen location model determines the functionality supported by a location service. Later it became evident that the location model also has major architectural implications.

Hierarchical symbolic locations are a recurring theme of this thesis. We think that this is the only way to successfully marry generality and scalability. We have shown how geometric location information can be incorporated into this model. Hierarchical data naturally lends itself to multi-resolution processing, which can be used to address scalability and performance problems.

Finally, our location model has been formally specified in Z, and also been implemented on an object-relational database platform.

8.1.3 Architectural approach

We have examined in detail the architecture of today's most widely used directory and location services, and found that some degree of replication is necessary in order to provide scalability. However, location information is too volatile for traditional replication. So the solution must be to replicate at the appropriate level of resolution. The previously-defined hierarchical location model can be employed to support such a scheme.

Further, we have identified partitioning by user and partitioning by location as important design approaches. User-agents and location-agents are the respective extreme cases. It appears to be impossible to find one partitioning scheme that is optimal in general. We can either chose a particular partitioning scheme based on a very narrow set of requirements or tolerate heterogeneous sub-systems within the architecture. For a global and general location service, only the second solution is viable.

We think that a general, global location service should be provided by a community of competing location service providers. Inter-working between service providers must be supported by a logically-centralised infrastructure, which offers directory services for locations and located-objects, as well as a scalable event propagation service.

8.1.4 Acquisition of location data

In many ways, acquiring and integrating location data from a set of heterogeneous location sensors is the core functionality of a support platform for location awareness. Acquisition consists of reception, abstraction, and fusion. Reception gathers data from sensors, abstraction translates it into a common representation, while fusion integrates multiple sightings of the same object.

A hierarchical data model facilitates abstraction and fusion. Then, the relationships between the translated locations become explicit so that conflicts and overlaps can be easily detected. We have proposed and specified a fusion algorithm over a location hierarchy. In our opinion, this demonstrates the superiority of a well-specified location model over an ad-hoc solution.

8.1.5 Uncertainty and prediction

Due to limited spatio-temporal resolution and other factors, location information is never perfect. This imperfection manifests itself as incorrect and incomplete information. Perhaps the most important limitation is that location sensors only provide information about the past. No element in the processing chain can be shielded completely from this fundamental uncertainty.

However, imperfection can be compensated for by injecting probabilistic external knowledge in the shape of prediction heuristics and mobility patterns. In order to know which heuristic or pattern is applicable, the quality of the trace must be measured. We have proposed continuity and directionality as abstract properties to be used for this task.

For movement prediction, a variety of mathematical models are available: Markov processes, statistical profiles, state machines, and others. We believe that it is important to match a trace to the right model. Also, one has to be aware that any prediction is inherently unreliable.

We have found that multi-resolution processing is an essential pre-requisite for serious location prediction. Hence, our hierarchical location model facilitates prediction. However, the orthogonal notions of adjacency, distance and directionality need to be added in order to reason about the movement of objects. Hence, we have proposed the concept of a *movement plane* consisting of a finite set of mutually exclusive locations. This model enables state-based reasoning about mobility.

8.1.6 Security and privacy

Many people do not like the prospect that other people might be able to track their every move. This concern is legitimate, even more so since somebody's activities can often be inferred from where they are and who they are with. Hence, mechanisms for protecting privacy of people and organisations are necessary. Since more protection implies less information (and thus functionality), such mechanisms must be flexible enough to strike an acceptable balance for most environments.

We have come to the conclusion that traditional protection models are not suitable for location information. This is because we can either specify how to protect a location or a located-object, but not both at the same time. So we have proposed multi-target access control to overcome this fundamental problem.

Also, location privacy is more about controlling the level of detail provided to clients rather than about a decision whether to allow or deny access. We have used the aforementioned multi-target access rules to specify policies for protection of location detail and anonymity. This is another instance of multi-resolution processing enabled by hierarchical locations.

8.1.7 Prototyping

We have gained a degree of confidence in our models with the support of two prototype implementations. A first prototype, an "Active Office" location service (cf. § 5.8, page 106), has allowed us to experiment with heterogeneous location sensors in a distributed environment. The acquisition framework described in chapter 5 has been implemented there. We have used the data gathered to perform statistical experiments in order to look for applicable prediction metrics and heuristics. Also, the implementation has provided initial validation for our architectural approach.

From this prototype, we have learned the following lessons:

- A clearly defined location model is essential when fusing data from heterogeneous sources.
- Client-side fusion is feasible and desirable, but conflicts with asynchronous location event notification.

- Inductive prediction is only feasible with high-quality location traces.

A second prototype, a database-centric location service (cf. appendix D), served as a testing platform for the hierarchic semi-symbolic location model (i.e. location domain model). Location queries and updates can be submitted in either geometric or symbolic form, thus reflecting the dual nature of the underlying location model. On this basis, we have implemented the protection model over the location hierarchy (cf. chapter 7).

We have learned the following lessons from this implementation:

- Conceptually, a location service is a database and can be implemented as such.
- Hierarchical structures and corresponding recursive operations do not perform very well in a relational database.
- Conventional database technology does not provide the scalability and distribution that is necessary for a global location service.

The two prototypes implement different aspects of our work. Together, they support the important concepts described in this thesis. However, there can be no doubt that a comprehensive, integral implementation of our ideas will lead to further insights.

8.2 Future Work

There is considerable scope for refining, extending, and applying the ideas in this thesis.

Scalable event propagation In the scenario of a global location service, one can easily imagine that there would be located-objects or locations which would be of interest to many people at the same time. Propagating updates via point-to-point notifications does not seem a viable strategy in such cases.

Therefore, we envisage that a global location service would be supported by a hierarchy of event channels propagating a restricted event set. Here, one could exploit a hierarchic location model as a hierarchic code to reduce traffic volume. Also, the event-channel hierarchy should be sparse, i.e. only required channels should exist. Further, low-level groupcast protocols should be used if an event channel has a large number of subscribers.

Security policies and implementation In the case of a location service spanning multiple administrative domains (analogous to today's Internet), traditional ways of authorising access seem less applicable. For example, the approach described in this thesis assumes that users are grouped into domains. But how are access rights assigned appropriately if the service is queried by somebody it has never met before?

We believe that novel models for establishing trust are needed for a large-scale open location service. Above all, such a model must be scalable to very large user populations.

Trusted location information While location information is often more or less uncertain, there are some potential application domains for location information where strong authenticity guarantees are required. For instance, a global organisation might (for legal reasons) only allow access to certain documents if the inquirer is not located in the United States of America. Here, reliable knowledge of the inquirer's location could be invaluable.

The thesis has pointed out some of the difficulties of generating reliable location information. It is a challenge to find a schema that works for multiple location sensor technologies!

Patterns for location-awareness Currently, the research community appears to struggle to develop viable paradigms for building location-aware applications and services. A set of commonly-recognised abstractions and design patterns would reduce the cost of developing the growing number of increasingly sophisticated location-aware applications.

While some of the models described in this thesis could be used as such, there is a need for more specific design patterns for location-awareness. For example, when should a component in a design expose location-awareness instead of location-transparency? What are the typical interactions?

Naturally, this list does not claim to be complete as it is biased by the preferences of the author. There are other areas, such as movement prediction heuristics or visualisation of location information which are also worthy of investigation.

8.3 Closing Remarks

Parts of this work and direct influences on it have been previously published:

Regis The Regis system [41] provided an implementation platform for most of this research.

Location Service The genesis of this thesis has been work on the Active Badge system by the author. A subsequent design by Jeff Magee [41] provided an invaluable conceptual and physical platform to start this research. Our first ideas were published in [34].

Security Policies For many years, Morris Sloman has headed research into system management at Imperial College. Our approach to security is based on this work. A comprehensive overview is can be found in [64]. Our work on access control for location information has been published in [35].

Appendix A

Glossary

Active Badge. An electronic tag developed by Olivetti that periodically sends out infrared beacon signals carrying its identity [74].

Agent. A software entity who acts for, or manages the transactions of, another.

Agent mobility. Refers to the ability of a software agent to migrate to another logical location (domain) and/or physical location (host) while executing.

Availability. Avoiding the denial of service [7].

Base station. The fixed transmitter/receiver device with which a mobile radio transceiver establishes a communication link to gain access to the public-switched telephone network [61].

Cell. The geographic area served by a single low-power transmitter/receiver. A cellular system's service area is divided into multiple cells [61].

Cellular connectivity. Communication provided by a radio or IR network consists of many cells (macro-cells, micro-cells, pico-cells) such that the moving mobile host frequently crosses cell boundaries.

Context-aware computing. The ability of a mobile user's applications to discover and react to changes in the environment in which they are situated [59].

Disconnected working. Continued operation without connection to the backbone network (e.g. vital servers).

Discretionary Access Control. A protection model which entails that the owner of an object has the discretion to allow or deny access to it.

GIS (Geographical Information System). Typically, a database-centred system for handling spatially-referenced static data.

GPS (Global Positioning System). Satellite-based radio navigation system [85].

GSM (Global System for Mobile Communications, originally Group Speciale Mobile). International standard for second-generation digital cellular mobile communication systems [48]. Currently, GSM-based systems are widely used in Europe and Asia.

HLR (Home Location Register). The logically centralised user profile database of a cellular communication system.

Horizontal application. A domain-independent software application, e.g. a spreadsheet [23].

Host mobility. The ability of a host to be physically moved to a new environment without significantly impeding its functionality.

Integrity. Maintaining integrity means preventing/detecting/deterring the improper modification of information [7].

Location domain. A well-defined geographical area providing a logical location for located-objects.

Location management. Gathering, storing, and disseminating information about the location of entities.

Location sensor. A sensing device whose inputs are used to measure the locations of objects.

Location service. A service that provides information about the physical location of real-world objects.

Location-awareness. The ability to adapt behaviour to the physical locations of users, resources, and processes.

Location-dependence. At different locations a different behaviour can be observed. This is not necessarily the result of location-awareness.

Located-object. A mobile object whose location can be tracked.

Location-sensitive information. The information about services or resources (including hardware and software resources, network connectivity, available communication protocols, etc.) provided by the system or networks in a defined location (i.e., geographical area) [39, 38].

Mandatory access control. A protection model which entails that access control is imposed by the system. Usually, this involves the labelling of documents and users. It is typically used to secure large amounts of information requiring strong protection in environments where system data can be classified and users cleared [7].

Mobile application. Software application which is partially or totally mobile. This mobility can be based on host or software mobility.

Mobile host. Abstraction for a computer that can be moved easily over a significant distance.

Mobility. The ability to move (or be moved).

Personal mobility. The mobility of a person between terminals and networks [38].

PCS (Personal Communication Services). A loosely-defined future ubiquitous telecommunications service that will allow "anytime, anywhere" voice and data communication with personal communication devices [61].

Privacy. "The right of an individual group or institution to determine when, how, and for what purpose information concerning himself/itself can be collected, stored, and released to other people or entities" [7].

QoS (Quality of Service). Quantitative property specification of service provisions or requirements. Examples: delay, throughput, jitter for communication channels (see [22]).

R-tree. A height-balanced tree, similar to a B-tree [5], which may be applied to index a large collection of points in multi-dimensional space [18].

Resource Mobility. Movement of resources (such as system data/programs, user data, user programs, etc.) in the underlying network to meet the QoS requirements of the mobile user [38, 39].

Roaming. A mobile user who roams temporarily uses a different communication service provider. Roaming typically happens when the mobile user is outside his or her home subscription area.

Secrecy. Preventing/detecting/deterring the improper disclosure of information [7].

Service Mobility. Movement of service logic in the underlying network to meet QoS requirements of the mobile user [38, 39]. A sub-category of resource mobility.

Terminal mobility. The mobility of a terminal within a mobile network system [38].

Terminal model. Design approach where the mobile host has the processing capabilities of a terminal, with applications running on a server computer.

User mobility. The user's ability to change the physical location where he or she accesses computing and communication services.

Vertical applications. Application for a specific application domain [23]

VLR (Visitor Location Register). Database temporarily storing subscription data for those subscribers currently within the service area of the corresponding switching centre of a cellular communication system [48].

Wireless connectivity. Channel to a communications network that does not rely on wire or fiber. Typically, based on radio-field or infrared signals.

WLAN (Wireless local area network). A computer network that allows transfer of data and provides the ability to share resources, such as printers, without the need to physically connect each node. WLANs may also offer mobility within an office or similar environment [61]. An example is Lucent's WaveLan.

Workstation model. Design approach where the mobile computer has the capabilities of a workstation, and will therefore run most applications locally.

Appendix B

A brief tour of Z

This appendix aims to provide the reader with a basic understanding of Z which will help to read the specifications given in this thesis. For a more comprehensive introduction we would like to refer the reader to [52].

Z is a specification language based on first-order logic and typed set theory. Specifications are divided into reusable schemas which can be composed into larger schemas. A specification consists of both formal and English components, providing a model of an actual system.

The English component consists of a rationale, a short description of the schema's meaning. In this thesis, this is provided by the text where the schema is embedded.

The formal component is represented by an open frame with a title and two sections: Signature and Predicate. The signature section consists of a list of variable declaration. It can be thought of as declaring the possible states of an object or operation conforming to the schema. The predicate section specifies constraints of the declared variables, for example state invariants, preconditions and postconditions of state transitions. Constraints are expressed using typed first-order predicate logic. Be aware that there is a rich set of pre-defined symbols and functions which can be used here!

Below is the formal component of a simple example schema:

Store

$$\textbf{contents} : \textbf{NUM}$$
$$\textbf{capacity} : \textbf{NUM}$$

$$\textbf{contents} \leq \textbf{capacity}$$

This schema declares an "object" named **Store** whose state consists of two variable: **contents** and **capacity**. The predicate section specifies a state invariant: The **contents** counter must be less than or equal to the **capacity** counter.

The above schema refers to the opaque type **NUM**. Normally, such types must be declared before use, for example by writing:

$$[\textbf{NUM}]$$

Further, Z contains a schema calculus, "which allows us to express a schema by making reference to other already defined schemas" ([53], page 140). In this thesis, we have mainly used schema inclusion which comes in three different flavours:

Inclusion declares that the including schema uses all the declarations and constraints of the included schema. For example:

NonEmptyStore

$$\textbf{Store}$$
$$\textbf{lowerBound} : \textbf{NUM}$$
$$\textbf{lowerBound} \leq \textbf{contents}$$

The resulting schema differs from **Store** by having an additional variable and an additional constraint.

Delta schema inclusion is used to indicate that a state transition is performed. For example:

FillStore

$$\Delta\textbf{Store}$$
$$\textbf{contents}' = \textbf{capacity}$$

This schema specifies a change to the state of **Store**. Additionally, we have added a post-condition expressing that the **Store** should be filled to its capacity.

Xi schema inclusion indicates that no state transition concerning the included schema was performed. For example:

MonitorStore

$$\Xi\textbf{Store}$$
$$\textbf{currentCount}? : \textbf{NUM}$$
$$\textbf{currentCount}? = \textbf{contents}$$

Here, Xi-inclusion of **Store** shows that the operation does not affect the state of **Store**. Note that the question mark at the end of **currentCount** indicates that it is an output variable.

Delta-inclusion and Xi-inclusion are actually defined as inclusions of implicitly defined Delta and Xi schemas, respectively. In Z, schema inclusion is a mechanism for building modular and well-structured specifications.

Finally, we have used generic schema definitions (templates) in this thesis. A template is simply a schema with one or more type variables. For example:

StoreTemplate[X]

$$\begin{array}{|l}
\textbf{contents} : \textbf{X} \\
\textbf{capacity} : \textbf{X} \\
\hline
\textbf{contents} \leq \textbf{capacity} \\
\hline
\end{array}$$

A template is instantiated with a defined type. In our example, **StoreTemplate[NUM]** is equivalent to **Store**.

Appendix C

Location services specified

This appendix provides specifications for symbolic, geometric, and hybrid location services. Parts of the specifications have been used in chapters 3 and 4. We use the Z formalism [52], combined with the Z/Eves specification checker [45].

C.1 A purely symbolic location service

This section specifies a location service purely based on hierarchical symbolic locations. We model this hierarchy as a partial ordering $<$ over symbolic locations. Located-objects can join and leave these locations freely. However, conflicting locations for the same object are disallowed, that is, all locations of a single object need to be on the same path through the hierarchy.

We start with the declaring the types of objects we deal with:

$$[\mathbf{LOCATION}, \mathbf{OBJECT}]$$

The inclusion ordering must be asymmetric and transitive:

IrreflexivePartialOrder[**X**]

$$\_ < \_ : \mathbf{X} \leftrightarrow \mathbf{X}$$

$$\forall \mathbf{x}, \mathbf{y} : \mathbf{X} \bullet$$
$$\quad \mathbf{x} < \mathbf{y} \Rightarrow \neg\, \mathbf{y} < \mathbf{x}$$
$$\forall \mathbf{x}, \mathbf{y}, \mathbf{z} : \mathbf{X} \bullet$$
$$\quad \mathbf{x} < \mathbf{y} \land \mathbf{y} < \mathbf{z} \Rightarrow \mathbf{x} < \mathbf{z}$$

The above template is instantiated below in a schema specifying a location hierarchy. The schema also contains provision for a root location (**anyLoc**), and a relationship associating non-overlapping locations (**conflicts**).

LocationHierarchy

$$\begin{array}{l}
\textbf{IrreflexivePartialOrder}[\textbf{LOCATION}] \\
\textbf{anyLoc} : \textbf{LOCATION} \\
\textbf{conflicts} : \textbf{LOCATION} \leftrightarrow \textbf{LOCATION} \\
\hline
\forall l1, l2 : \textbf{LOCATION} \bullet \\
\quad (l1, l2) \in \textbf{conflicts} \Leftrightarrow \\
\quad\quad (\forall l3 : \textbf{LOCATION} \bullet \\
\quad\quad\quad \neg ((l3 < l1 \lor l3 = l1) \land (l3 < l2 \lor l3 = l2))) \\
\forall l : \textbf{LOCATION} \bullet \\
\quad l < \textbf{anyLoc} \lor l = \textbf{anyLoc}
\end{array}$$

SymbolicLocator

$$\begin{array}{l}
\textbf{locatedAt} : \textbf{OBJECT} \leftrightarrow \textbf{LOCATION}
\end{array}$$

For a location service, we assert that no conflicting locations are allowed for any located-object. More than one location per object is permitted.

SymbolicLocationService

$$\begin{array}{l}
\textbf{LocationHierarchy} \\
\textbf{SymbolicLocator} \\
\hline
\forall \mathbf{o} : \textbf{OBJECT}; l1, l2 : \textbf{LOCATION} \bullet \\
\quad (\mathbf{o}, l1) \in \textbf{locatedAt} \land (\mathbf{o}, l2) \in \textbf{locatedAt}) \Rightarrow (l1, l2) \notin \textbf{conflicts}
\end{array}$$

Firstly, a simple query for all located-objects at a given location:

SLocationQuery

$$\begin{array}{l}
\Xi \textbf{SymbolicLocationService} \\
\textbf{target?} : \textbf{LOCATION} \\
\textbf{result!} : \mathbb{P}\, \textbf{OBJECT} \\
\hline
\textbf{result!} = \{\mathbf{x} : \textbf{OBJECT} \mid (\mathbf{x}, \textbf{target?}) \in \textbf{locatedAt}\}
\end{array}$$

Then, a recursive query that includes sub-locations:

SLocationQueryComplete

$$\begin{array}{l}
\Xi \textbf{SymbolicLocationService} \\
\textbf{target?} : \textbf{LOCATION} \\
\textbf{result!} : \mathbb{P}\, \textbf{OBJECT} \\
\hline
\textbf{result!} = \{\mathbf{x} : \textbf{OBJECT} \mid (\mathbf{x}, \textbf{target?}) \in \textbf{locatedAt} \lor \\
\quad (\exists \mathbf{y} : \textbf{LOCATION} \bullet \mathbf{y} < \textbf{target?} \land (\mathbf{x}, \mathbf{y}) \in \textbf{locatedAt})\}
\end{array}$$

This query returns all locations at which a located-object is positioned:

SObjectQuery

$$\Xi \mathbf{SymbolicLocationService}$$
$$\mathbf{target?} : \mathbf{OBJECT}$$
$$\mathbf{result!} : \mathbb{P}\,\mathbf{LOCATION}$$

$$\mathbf{result!} = \{\mathbf{x} : \mathbf{LOCATION} \mid (\mathbf{target?}, \mathbf{x}) \in \mathbf{locatedAt}\}$$

The same as above, but now including all super-locations:

SObjectQueryComplete

$$\Xi \mathbf{SymbolicLocationService}$$
$$\mathbf{target?} : \mathbf{OBJECT}$$
$$\mathbf{result!} : \mathbb{P}\,\mathbf{LOCATION}$$

$$\mathbf{result!} = \{\mathbf{x} : \mathbf{LOCATION} \mid (\mathbf{target?}, \mathbf{x}) \in \mathbf{locatedAt} \lor (\exists\, \mathbf{y} : \mathbf{LOCATION} \bullet \mathbf{y} < \mathbf{x} \land (\mathbf{target?}, \mathbf{y}) \in \mathbf{locatedAt})\}$$

A sighting is inserted by removing previous locations of the object and adding the new location:

SUpdate

$$\Xi \mathbf{LocationHierarchy}$$
$$\Delta \mathbf{SymbolicLocator}$$
$$\mathbf{l?} : \mathbf{LOCATION}$$
$$\mathbf{o?} : \mathbf{OBJECT}$$

$$\mathbf{locatedAt}' = (\{\mathbf{o?}\} \mathbin{\not\Leftrightarrow} \mathbf{locatedAt}) \cup \{(\mathbf{o?}, \mathbf{l?})\}$$

Note that explicit updates may not be necessary if **locatedAt** maps to an on-demand measurement of the located-object's position (for example, by performing a radio triangulation).

C.2 A geometric location service

A purely geometric location service deals with located-objects and their geometric positions. The sort of located-object has been defined above. The geometric categories of interest are:

$$[\mathbf{AREA}, \mathbf{POINT}]$$

A geometric ordering relation asserting asymmetry and transitivity:

$$\mathbf{contains} : \mathbf{AREA} \leftrightarrow \mathbf{AREA}$$

$$\forall \mathbf{a}1, \mathbf{a}2 : \mathbf{AREA} \bullet (\mathbf{a}1, \mathbf{a}2) \in \mathbf{contains} \Rightarrow (\mathbf{a}2, \mathbf{a}1) \notin \mathbf{contains}$$
$$\forall \mathbf{a}1, \mathbf{a}2, \mathbf{a}3 : \mathbf{AREA} \bullet$$
$$(\mathbf{a}1, \mathbf{a}2) \in \mathbf{contains} \land (\mathbf{a}2, \mathbf{a}3) \in \mathbf{contains} \Rightarrow (\mathbf{a}1, \mathbf{a}3) \in \mathbf{contains}$$

For convenience:

$$\mathbf{containsEq} : \mathbf{AREA} \leftrightarrow \mathbf{AREA}$$

$$\forall \mathbf{a}1, \mathbf{a}2 : \mathbf{AREA} \bullet (\mathbf{a}1, \mathbf{a}2) \in \mathbf{containsEq} \Leftrightarrow (\mathbf{a}1, \mathbf{a}2) \in \mathbf{contains} \lor \mathbf{a}1 = \mathbf{a}2$$

We prefer to deal only with areas because points naturally translate into areas.

$$\mathbf{asArea} : \mathbf{POINT} \rightarrow \mathbf{AREA}$$

$$\forall \mathbf{p}1, \mathbf{p}2 : \mathbf{POINT} \bullet \neg (\mathbf{p}1 = \mathbf{p}2) \Rightarrow \neg (\mathbf{asArea}(\mathbf{p}1) = \mathbf{asArea}(\mathbf{p}2))$$

Located-objects *can* have a geometric position.

GeometricLocator

$$\mathbf{position} : \mathbf{OBJECT} \nrightarrow \mathbf{AREA}$$

GeometricLocationService

$$\mathbf{GeometricLocator}$$

This query returns all located-objects within a geographical area:

GLocationQuery

$$\Xi \mathbf{GeometricLocationService}$$
$$\mathbf{target?} : \mathbf{AREA}$$
$$\mathbf{result!} : \mathbb{P}\, \mathbf{OBJECT}$$

$$\mathbf{result!} = \{\mathbf{x} : \mathbf{OBJECT} \mid (\mathbf{target?}, \mathbf{position}(\mathbf{x})) \in \mathbf{containsEq}\}$$

This query returns a located-object's geometric position, which may be unknown (modelled by an empty set):

GObjectQuery

$$\Xi \mathbf{GeometricLocationService}$$
$$\mathbf{target?} : \mathbf{OBJECT}$$
$$\mathbf{result!} : \mathbb{P}\, \mathbf{AREA}$$

$$\mathbf{result!} = \{\mathbf{position}(\mathbf{target?})\}$$

An update overwrites any information about previous positions:

GUpdate

Δ**GeometricLocationService**
a? : **AREA**
o? : **OBJECT**

---

position$'$ = ({**o**?} $\lhd$ **position**) $\cup$ {(**o**?, **a**?)}

C.3 A hybrid location service

This section describes how geometric data can be combined with the purely symbolic location model specified above. This is achieved by mapping locations into areas such that location ordering is isomorphic to spatial containment of areas.

Over the combined, semi-symbolic location model we define the operations previously specified for symbolic and geometric models. We extend **SymbolicLocator** by assigning an area to each location. Further, we require that the **area** function defines an isomorphism with regard to the ordering relations. The **leastMatchingLoc** function returns the smallest location that contains or is equal to a given area.

HLocationHierarchy

LocationHierarchy
area : **LOCATION** $\rightarrow$ **AREA**
leastMatchingLoc : **AREA** $\nrightarrow$ **LOCATION**

$\forall$ l1, l2 : **LOCATION** $\bullet$
  (l1 $<$ l2) $\Leftrightarrow$ (**area**(l2), **area**(l1)) $\in$ **contains**
$\forall$ **a** : **AREA**; l3 : **LOCATION** $\bullet$
  leastMatchingLoc(**a**) = l3 $\Leftrightarrow$
    (**area**(l3), **a**) $\in$ **containsEq** $\wedge$
    ($\forall$ l4 : **LOCATION** $\bullet$
      (**area**(l4), **a**) $\in$ **containsEq** $\Rightarrow$ ((l3 = l4) $\vee$ (l3 $<$ l4)))

We assert that locations stored symbolically are consistent with **SymbolicLocator**. Further, we require that locations stored symbolically reflect the position stored in **GeometricLocator** as accurately as possible.

HLocationService

$$
\begin{array}{l}
\mathbf{HLocationHierarchy} \\
\mathbf{SymbolicLocator} \\
\mathbf{GeometricLocator} \\
\hline
\forall\, \mathbf{o} : \mathbf{OBJECT}; \mathrm{l} : \mathbf{LOCATION} \bullet \\
\quad (\mathbf{o} \in \mathrm{dom}\, \mathbf{locatedAt}) \Rightarrow (\mathbf{o} \in \mathrm{dom}\, \mathbf{position}) \\
\forall\, \mathbf{o} : \mathbf{OBJECT}; \mathrm{l} : \mathbf{LOCATION} \bullet \\
\quad (\mathbf{o}, \mathrm{l}) \in \mathbf{locatedAt} \Rightarrow (\mathbf{area}(\mathrm{l}), \mathbf{position}(\mathbf{o})) \in \mathbf{containsEq} \\
\forall\, \mathbf{o} : \mathbf{OBJECT}; \mathrm{l1}, \mathrm{l2} : \mathbf{LOCATION} \bullet \\
\quad ((\mathbf{o} \in \mathrm{dom}\, \mathbf{position}) \land \\
\quad (\mathbf{area}(\mathrm{l1}), \mathbf{position}(\mathbf{o})) \in \mathbf{containsEq} \land \\
\quad (\mathbf{o}, \mathrm{l2}) \in \mathbf{locatedAt}) \Rightarrow \\
\qquad ((\mathrm{l1} = \mathrm{l2}) \lor (\mathrm{l2} < \mathrm{l1}))
\end{array}
$$

The following schemas define operations over a hybrid location service. Each type of operation exists in two variants: a geometric and a symbolic version.

This query finds all located-objects at a given symbolic location:

HSLocationQuery

$$
\begin{array}{l}
\Xi \mathbf{HLocationService} \\
\mathbf{target?} : \mathbf{LOCATION} \\
\mathbf{result!} : \mathbb{P}\, \mathbf{OBJECT} \\
\hline
\mathbf{result!} = \{\mathbf{x} : \mathbf{OBJECT} \mid (\mathbf{x}, \mathbf{target?}) \in \mathbf{locatedAt} \lor \\
\quad (\exists\, \mathbf{y} : \mathbf{LOCATION} \bullet (\mathbf{y} < \mathbf{target?}) \land (\mathbf{x}, \mathbf{y}) \in \mathbf{locatedAt})\}
\end{array}
$$

This query finds all located-objects in a given geometric area:

HGLocationQuery

$$
\begin{array}{l}
\Xi \mathbf{HLocationService} \\
\mathbf{target?} : \mathbf{AREA} \\
\mathbf{result!} : \mathbb{P}\, \mathbf{OBJECT} \\
\hline
\mathbf{result!} = \{\mathbf{x} : \mathbf{OBJECT} \mid \\
\quad (\mathbf{x}, \mathbf{leastMatchingLoc}(\mathbf{target?})) \in \mathbf{locatedAt} \land \\
\quad (\mathbf{target?}, \mathbf{position}(\mathbf{x})) \in \mathbf{containsEq}\}
\end{array}
$$

This query returns all symbolic locations associated with a particular located-object:

HSObjectQuery

Ξ**HLocationService**
target? : **OBJECT**
result! : $\mathbb{P}$ **LOCATION**

$$\textbf{result!} = \{\textbf{x} : \textbf{LOCATION} \mid (\textbf{target?}, \textbf{x}) \in \textbf{locatedAt} \lor (\exists\, \textbf{y} : \textbf{LOCATION} \bullet (\textbf{y} < \textbf{x}) \land (\textbf{target?}, \textbf{y}) \in \textbf{locatedAt})\}$$

This query returns a singleton set with the current geometric location of the queried located-object:

HGObjectQuery

Ξ**HLocationService**
target? : **OBJECT**
result! : $\mathbb{P}$ **AREA**

$$\textbf{result!} = \{\textbf{position}(\textbf{target?})\}$$

This operation updates the located-object's position with a new geometric location:

HGUpdate

Δ**HLocationService**
a? : **AREA**
o? : **OBJECT**

$$\textbf{position}' = (\{\textbf{o?}\} \lhd \textbf{position}) \cup \{(\textbf{o?}, \textbf{a?})\}$$
$$\textbf{locatedAt}' = (\{\textbf{o?}\} \lhd \textbf{locatedAt}) \cup \{(\textbf{o?}, \textbf{leastMatchingLoc}(\textbf{a?}))\}$$

This operation updates a located-object's position with a new symbolic location:

HSUpdate

Δ**HLocationService**
l? : **LOCATION**
o? : **OBJECT**

$$\textbf{position}' = \textbf{position} \oplus \{(\textbf{o?} \mapsto \textbf{area}(\textbf{l?}))\}$$
$$\textbf{locatedAt}' = (\{\textbf{o?}\} \lhd \textbf{locatedAt}) \cup \{(\textbf{o?}, \textbf{l?})\}$$

Appendix D

A database-centric general location service

This thesis has identified the models and abstractions on which a general location service should be based. A prototype implementation uses an object-relational database management system with a plug-in for three-dimensional spatial data-types and operations (Informix' Illustra with 3-D spatial DataBlade [24, 25]).

The implementation supports the semi-symbolic location model (cf. chapter 3), the service model (cf. § 4.3)), and the protection model (cf. chapter 7) proposed in this thesis. We use the server implementations described in § 5.8 to acquire location sightings. Additionally, inputs from GPS receivers are used.

D.1 Overview

The database schema consists of two main tables, one for records of located-objects and one for locations. Object-records hold all the status associated with a particular user, while location-records hold spatial information for a particular symbolic location. Additionally, alias-tables for both user and location names provide a many-to-one mapping. This is useful, for example, when mapping a user's badge identifier and email address to the same user-record. Further, we have three rule tables specifying access control: Accessibility, Visibility, and Anonymity. These are employed to filter query results.

D.2 Representation of located-object information

```
create table LObjects of new type object_t (
    Id userid_t not null primary key,
    CPos Poly3d, /* model users position as area of uncertainty */
    TOS Timestamp /* time of sighting*/
);
```

The located-object record contains the unique identifier of the user (currently the email address), along with the time and coordinates of last sighting. More sophisticated trace information could also be stored here.

The position of a located-object is modelled as a geometric area (not as a point) in order to allow for uncertainty in updates and query results.

D.3 Representation of location information

```
create table Locations of new type location_t (
    Id locid_t  not null primary key,
    Boundary Poly3d,
    SubAreas setof(locid_t),
    When Timestamp /* last update */
);
```

Locations are identified by a globally unique identifier and the geometric area enclosed by Boundary. Alternatively, a location can consist of a set of SubAreas. Since locations may change position, we also record the time of last update.

Physically, location records are stored in a flat table. Logically, however, they form a lattice defined by the containment relation. Most queries and updates operate on this conceptual lattice.

In our schema, the lattice is implicit and needs to be computed whenever we need to traverse it. In terms of performance, this is its most significant drawback.

D.4 Access control

```
create table AccessR of new type rule_t (
    Suid    userid_t, /* subject */
    Tuid    userid_t, /* target user */
    Lid locid_t       /* target location */
);

create table VisiR of type rule_t;
create table AnonR of type rule_t;
```

Access rules Access rules define a geographical area of accessibility for each pair of (source user, target user). They are the first restriction applied to a query. If a user is queried, and the user is outside any authorised geographical area for that (source user, target user) pair, the query is rejected. If a location is queried, all users that are not accessible are removed from the result set.

Visibility rules Visibility rules define a set of visible symbolic locations for each pair of (source user, target user). If a location is queried, users that are not visible there are removed from the result set. If a user is queried, all locations where the user is not supposed to be visible are removed from the results set.

Anonymity rules Anonymity rules specify for each triple (source user, target user, symbolic location) whether the source user can identify the target when located at the given location. If a location is queried, the identity of all users in the result set that are not allowed to be identified is changed to 'Anonymous'. If a user is queried, 'Anonymous' locations (i.e. those where the target is not allowed to be identified) are stripped from the result set.

D.5 Updates

D.5.1 Geometric sightings

```
create function loc_sighting_poly(useral_t,Poly3d, Timestamp)
returns void
as
begin
    update LObjects
    set
        CPos = $2,
        TOS = $3
    where
        LObjects.Id = get_uid($1)
        AND ( LObjects.TOS IS NULL  OR LObjects.TOS < $3);
end;
```

We model geometric sightings as a triple (Object Alias, Area, Timestamp). In terms of processing, we look up the relevant object records and update the current positions if the new sighting is more recent then the last recorded sighting.

D.5.2 Symbolic sightings

```
create function loc_sighting(userid_t,locid_t,Timestamp)
returns void
as
begin
    update LObjects
    set
        CPos =
            (select unique Boundary
            from Locations
            where  Locations.Id = $2),
        TOS = $3
    where LObjects.Id = $1 AND LObjects.TOS < $3;
end;
```

Firstly, we resolve the location and object aliases. Then, we update the object's current position with the area corresponding to the specified symbolic location. (Naturally, this

update is conditional on the new sighting being more recent then the previous position in the user record.)

D.6 Queries

D.6.1 Symbolic queries

```
create type symb_sighting_t
(
    Id id_t,
    TOS Timestamp
);
```

In our model, symbolic queries return a set of sightings relative to the target of the query, i.e. the target itself is not included in the sightings.

```
/* return set of users for a given location alias */
create function get_users_alias (local_t)
returns setof(symb_sighting_t)
as
    select symb_sighting_t(LObjects.Id,LObjects.TOS)
        from Locations, LObjects
        where get_lid($1) = Locations.Id AND
                            ns_contains(Locations, LObjects);
```

This query looks up a location record and then scans through all objects to discover whether they happen to be in this area. The check `ns_contains()` is recursive if the location is composed of sub-areas.

By using an R-tree index [18] over the current positions of the located-objects we can avoid a linear table-scan but incur an increased update cost.

Alternatively, the location records could know about the objects currently within their area. (This solution would mean that updates are more costly.) Moreover, the update cost goes up with the number of locations to be updated.

The recursive containment check as performed by `ns_contains()` should probably be executed in a more efficient way outside the databases.

```
/* return set of locations for a given user id */
create function get_locs (userid_t)
returns setof(symb_sighting_t)
as
    select symb_sighting_t( l.Id , u.TOS)
        from Locations l, LObjects u
        where $1 = u.Id AND ns_contains(l, u);
```

This query is similar to the previous one, except that here the geometric area is the last position of the located-object. The points made about optimisation of the previous query apply.

For purposes of access control, we layer filters on top of the two basic queries described above.

```
create function is_accessible(useral_t,useral_t,Poly3d)
returns boolean
as
    return exists(select * from Locations l
        where ns_contains(l, $3)
            AND checkAccessRule(get_uid($1),get_uid($2),l.Id));

create function safe_query_user(useral_t, useral_t)
returns setof(symb_sighting_t)
as
    select * from  get_locs_alias($2)
             where is_accessible($1,$2,deref(get_user($2)).CPos);

create function safe_filtered_query_user(useral_t, useral_t)
returns setof(symb_sighting_t)
as
    select * from safe_query_user($1,$2) p where
        is_visible(get_uid($1),get_uid($2),p.Id::locid_t)
        AND
        is_identifiable(get_uid($1),get_uid($2),p.Id::locid_t);
```

In this example, `is_accessible()` performs a recursive rule lookup to find out whether access to the user's current position is allowed. The functions `is_visible()` and `is_identifiable()` perform straight lookups to the rule tables for visibility (`VisiR`) and anonymity (`AnonR`).

D.6.2 Geometric queries

Queries for located-objects within a specific area can be satisfied in the same way as above. As far as access control is concerned, only Accessibility protection applies. The same holds, if an object's current location is queried.

Queries of the type "Find the nearest..." can be satisfied by using temporary and successively enlarged locations with the above queries.

D.7 Critical evaluation

The choice of a database as a prototyping platform has allowed for an extremely short development time of the prototype. However, even for a relatively small data set (parts of our Computing department), queries and updates take on the order of seconds rather than milliseconds. Unsurprisingly, this loss of speed happens mostly when recursive queries are involved. In our model, the location hierarchy is the only recursive data structure. This should be implemented outside the database, while flat tables (such as user profiles or access rules) remain within the database.

Issues of distribution are also not adequately addressed by this prototype. For a real implementation, a distribution mechanism outside the database must be found. At the same time, it may still be beneficially to use local databases for tasks that they perform well.

D.8 Summary

We have implemented and verified the functions and abstractions for a general location service. Our prototype proved a useful workbench to develop this model further. In order to investigate performance and scalability issues it will be necessary to choose another implementation platform (at least for critical parts of the design.)

Appendix E

Spatial relationships

In this appendix, we give a brief account of our terminology for some of the fundamental spatial relationships.

E.1 Inclusion

This is treated as fundamental notion. The spatial concept that is being modelled is strict area inclusion (i.e. equality is excluded).

Notation: $\mathbf{a} < \mathbf{b}$ means that location $\mathbf{a}$ is a strict sub-area of $\mathbf{b}$.

Properties:

- Transitivity: $\mathbf{a} < \mathbf{b} \wedge \mathbf{b} < \mathbf{c} \Rightarrow \mathbf{a} < \mathbf{c}$
- Asymmetry: $\mathbf{a} < \mathbf{b} \Rightarrow \neg\, \mathbf{b} < \mathbf{a}$

Corollaries:

- Inclusion is a partial ordering relation.
- Inclusion is irreflexive.
- The least upper bound and greatest lower bound can be computed.

E.2 Overlap

This notion can be derived from inclusion. Definition: $\mathbf{overlaps}(\mathbf{a}, \mathbf{b})$ means that the locations $\mathbf{a}$ and $\mathbf{b}$ share a common sub-area: $\mathbf{overlaps}(\mathbf{a}, \mathbf{b}) \Leftrightarrow (\exists\, \mathbf{c}).(\mathbf{c} < \mathbf{a} \wedge \mathbf{c} < \mathbf{b})$

Corollaries:

- Overlaps are not transitive.
- Overlaps are symmetric.
- Overlaps are reflexive.
- An area overlaps with all its sub-areas.

E.3 Adjacency

This is a fundamental notion that models how located-objects traverse the location space. In order to keep adjacency independent from inclusion, we allow adjacent locations to overlap. Also, adjacency must be directional.

Definition: $\mathbf{adj}(\mathbf{a}, \mathbf{b})$ means that a located-object can travel from **a** to **b** without setting foot on another location.

Properties:

- Adjacency is not transitive
- Adjacency is not symmetric.
- Adjacency is reflexive
- Adjacency is implied by inclusion and overlap.

Appendix F

Properties of age-weighted metrics

F.1 Properties of aged average

The aged average has the following general form:

$$\begin{aligned} \widetilde{\mathrm{m}}(\mathbf{T}) &= \frac{\sum_{\mathbf{s}\in\mathbf{T}}(\mathrm{m}(\mathbf{s})\cdot age^{\mathbf{T}}(\mathbf{s}))}{\sum_{\mathbf{s}\in\mathbf{T}} age^{\mathbf{T}}(\mathbf{s})} \\ &= \frac{\sum_{\mathbf{s}\in\mathbf{T}}(\mathrm{m}(\mathbf{s})\cdot e^{-\mathbf{a}\cdot(\mathbf{t_e^T}-\mathbf{t^s})})}{\sum_{\mathbf{s}\in\mathbf{T}} e^{-\mathbf{a}\cdot(\mathbf{t_e^T}-\mathbf{t^s})}} \end{aligned}$$

Properties:

- Given an even event distribution, older events become increasingly insignificant for the computation of the average.
- If $\mathrm{m}(\mathbf{s}) = \mathbf{const}$, $\widetilde{\mathrm{m}}(\mathbf{T})$ equals the normal average($\overline{\mathrm{m}}(\mathbf{T})$).
- A trace starved of recent events will converge towards it normal average.
- The upper bound of $\widetilde{\mathrm{m}}(\mathbf{T})$ is $\mathbf{max}(\mathrm{m}(\mathbf{s}))$.

F.2 Properties of aged density

We have discussed measures of the general form:

$$\begin{aligned} \widetilde{\delta}_{\mathbf{m}}(\mathbf{T}) &= \sum_{\mathbf{s}\in\mathbf{T}}(\mathrm{m}(\mathbf{s})\cdot age^{\mathbf{T}}(\mathbf{s})) \\ &= \sum_{\mathbf{s}\in\mathbf{T}}(\mathrm{m}(\mathbf{s})\cdot e^{-\mathbf{a}\cdot(\mathbf{t_e^T}-\mathbf{t^s})}) \end{aligned}$$

Assuming that the event density is bounded (e.g. if trace $\mathbf{T}$ is finite), such measures $\widetilde{\delta}_{\mathbf{m}}(\mathbf{T})$ satisfy the following properties:

1. Events that are sufficiently old become insignificant for the computation of the measure. This allows us to ignore events earlier than a time $\mathbf{t_x}$ with $\mathbf{t_s^T} < \mathbf{t_x} < \mathbf{t_s^T}$ without unduly affecting the result. The choice of $\mathbf{t_x}$ depends on the constant $\mathbf{a}$, the maximum event density, and the required accuracy of the result.

2. The measures for a trace starved of recent events converge towards zero. This is a corollary of the previous point.

3. Given the event density is bounded, also the measure $\widetilde{\delta}_{\mathrm{m}}(\mathbf{T})$ is bounded.

Metrics of this form are particularly useful to compare competing traces. In isolation, the metric's value is useful only when an expected value is known. For example, see § 6.2.3 for some expected measures for Active Badges.

However, those metrics alone do not indicate how closely a sighting trace follows the located-object's movement. This is only possible when taking into account movement characteristics (especially the speed of the located-object.)

Bibliography

[1] *IEEE Communications*, 35(2), February 1997.

[2] ISO/IEC JTC/SC 21. Provision of Trading Function using OSI Directory Service, May 1996. Draft Recommendation X.950-3 (ODP–Trading Function–Part 3).

[3] B. Awerbuch and D. Pele. Concurrent online tracking of mobile users. In *Proceedings of the ACM SIGCOMM Conference on Communication Architectures and Protocols*, volume 2 of *Computer Communication Review*, pages 221–233, Zurich, Switzerland, September 1991. ACM Press.

[4] B. Badrinath, T. Imielinski, and A. Virmani. Locating strategies for personal communication networks. In *Proceedings of IEEE Globecom'92 Workshop on Networking for Personal Communications Application*, December 1992.

[5] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the ACM SIGFIDET Workshop on Data Description and Access*, pages 107–141, Houston, Texas, November 1970.

[6] M. Bearman. Tutorial on ODP Trading Function, February 1997. Available as http://www.dstc.edu.au/AU/research_news/odp/trader/tr_tutorial.html.

[7] S. Castano et al. *Database Security.* Addison-Wesley, 1994.

[8] D. Chadwick. *Understanding X.500 The Directory.* Chapman & Hall, 1994.

[9] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Security and Privacy Symposium*, pages 184–194, April 1987.

[10] S. Crane. *Dynamic Binding in Distributed Systems.* PhD thesis, Department of Computing, Imperial College, London, May 1997.

[11] D. Crocker. RFC 822. Standard for the format of ARPA Internet text messages, August 1982.

[12] J. Daintith and R. D. Nelson, editors. *The Penguin Dictionary of Mathematics.* Penguin Books, 1989.

[13] C. Drane and C. Rizos. *Positioning Systems in Intelligent Transportation Systems.* Intelligent Transportation Systems. Artech House, 1998.

[14] K. Duddy, K. Raymond, and A. Vogel. Trader down under: Upside down and inside out. Technical Report 25, CRC for Distributed Systems Technology, University of Queensland, Australia, 1995.

[15] G. Eleftheriadis and M. Theologou. User profile identification in future mobile telecommunications systems. *IEEE Network*, 8(5):33–39, September 1994.

[16] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3), June 1987.

[17] J. Geier. *Wireless Networking Handbook*. New Riders Publishing, 1996.

[18] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 13th ACM SIGMOD conference*, pages 47–57, Boston, 1984. ACM press.

[19] F. Halsall. *Data Communications, Computer Networks and Open Systems*. Addison-Wesley, fourth edition, 1995.

[20] A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network*, 36(1):62–70, January 1994.

[21] T. Howes and M. Smith. RFC 1823. The LDAP Application Program Interface, August 1995.

[22] D. Hutchison, G. Coulson, A. Campbell, and G. Blair. Quality of service management in distributed systems. In Sloman [64], pages 273–302.

[23] T. Imielinski and B. Badrinath. Mobile wireless computing. *Communications of the ACM*, 37(10):19–28, October 1994.

[24] Informix, Inc. *3D Spatial DataBlade Guide*, March 1995.

[25] Informix, Inc. *Illustra User's Guide*, October 1995.

[26] R. Jain. Reducing traffic impact of PCS using hierarchical user location databases. In *Proceedings of the International Communications Conference (ICC'96)*, Dallas, Texas, June 1996.

[27] R. Jain and N. Krishnakumar. Service handoffs and virtual mobility for delivery of personal information services to mobile users. Technical Memorandum TM-24696, Bell Communications Research, December 1994.

[28] R. Jain, Y.-B. Lin, and S. Mohan. Location strategies for personal communication services. In J. Gibson, editor, *Mobile Communications Handbook*. CRC press, May 1995.

[29] P. Krishna, N. Viadya, and D. Pradhan. Static and dynamic location management in distributed mobile environments. Technical report, Department of Computer Science, Texas A&M University, June 1994.

[30] T. La Porta, M. Veeraraghavan, P. Treventi, and R. Ramjee. Distributed call processing for personal communicaton services. *IEEE Communications Magazine*, 33(6):66–75, June 1995.

[31] D. Lam, D. Cox, and J. Widom. Teletraffic modeling for personal communications services. In *IEEE Communications* [1], pages 79–87.

[32] B. W. Lampson. Protection. In *Proceedings of the Fifth Annual Princeton Conference on Information Science Systems*, pages 437–443, 1971. Reprinted in Operating Systems Review, Volume 8, Number 1 (January 1974), pages 18–24.

[33] T. Lane. In M. Shaw and D. Garlan, editors, *Software Architecture*, chapter Architectural Design Guidance. Prentice Hall, 1996.

[34] U. Leonhardt and J. Magee. Towards a general location service for mobile environments. In *Proceedings of the Third International Workshop on Services in Distributed and Networked Environments*, pages 43–50, Macau, June 1996. IEEE CS Press.

[35] U. Leonhardt and J. Magee. Security considerations for a distributed location service. *Journal of Systems and Network Management*, 6(1):51–70, March 1998.

[36] G. Liu. Efficient mobility management for wireless mobile computing and communications. Licentiate's thesis, Telecommunications System Laboratory, Department of Teleinformatics, Royal Institute of Technology Stockholm, March 1995.

[37] G. Liu. *The Effectiveness of a Full-Mobility Architecture for Wireless Mobile Computing and Personal Communications.* PhD thesis, Telecommunications System Laboratory, Department of Teleinformatics, Royal Institute of Technology Stockholm, March 1996.

[38] G. Liu and G. Maguire. A virtual distributed system architecture for supporting global-distributed mobile computing. Technical Report TRITA-IT R 95-01, Department of Teleinformatics, Royal Institute of Technology, S-164 40 Kista, Sweden, December 1994.

[39] G. Liu and G. Maguire. Efficient mobility management support for wireless data services. In *Proceedings of the 45th IEEE Vehicular Technology Conference (VTC'95)*, Chicago, Illinois, July 1995. IEEE.

[40] H. Maaß. Location-aware mobile applications based on directory services. In *Proceedings of the Third International Conference on Mobile Computing and Networking (MOBICOM'97)*, pages 23–33, Budapest, Hungary, September 1997. ACM Press.

[41] J. Magee, N. Dulay, and J. Kramer. Regis: A constructive development environment for distributed programs. *Distributed Systems Engineering*, 5(1):304–312, September 1994.

[42] G. Maguire, F. Reichert, and M. Smith. A multiport mobile internet router. In *Proceedings of the 44th IEEE Vehicular Technology Conference*, Stockholm, Sweden, June 1994.

[43] G. Maguire, M. Smith, and T. Osawa. Walkstation II project. In *Proceedings of the Second International Workshop on Mobile Multi-Media Communications*, Bristol, UK, April 1995.

[44] D. Marriott and M. Sloman. Management policy service for distributed systems. In *Proceedings of the Third International Workshop on Services in Distributed and Networked Environments*, pages 2–9, Macau, June 1996. IEEE CS Press.

[45] I. Meisels and M. Saaltink. *The Z/EVES Reference Manual*. ORA Canada, December 1995.

[46] P. Mokapetris. RFC 1034. Domain Names — Concepts and Facilities, November 1987.

[47] P. Mokapetris. RFC 1035. Domain Names — Implementation and Specification, November 1987.

[48] M. Mouly and M.-B. Pautet. *The GSM System for Mobile Communications*. Published by the authors, 4 rue Elisée Reclus, F-91120 Palaiseau, France, 1992.

[49] R. Needham. Names. In S. Mullender, editor, *Distributed Systems*, ACM Press Frontier Series, chapter 12, pages 315–327. Addision-Wesley, second edition, 1993.

[50] C. Perkins. RFC 2002. IP Mobility Support, October 1996.

[51] C. Popien and B. Meyer. Federating ODP traders: An X.500 approach. In *Proceedings of the International Conference on Communications*, pages 313–317, Geneva, Switzerland, 1993.

[52] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice-Hall, 1991.

[53] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice-Hall, second edition, 1996.

[54] T. Richardson, F. Bennett, G. Mapp, and A. Hopper. Teleporting in an X Window System Environment. *IEEE Personal Communications Magazine*, 1(3):6–12, 1994.

[55] M. Rizzo, P. Linington, and I. Utting. Integration of location services in the Open Distributed Office. Technical Report 14-94, University of Kent, Computing Laboratory, Canterbury, UK, August 1994.

[56] C. Rose and R. Yates. Location uncertainty in mobile networks: A theoretical framework. In *IEEE Communications* [1], pages 94–101.

[57] J. Saltzer, D. Reed, and D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[58] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, December 1994.

[59] B. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, pages 22–32, September 1994.

[60] B. Schilit, M. Theimer, and B. Welch. Customizing mobile applications. In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pages 129–138, Cambridge, Massachusetts, August 1993.

[61] R. Schneiderman. *Wireless Personal Communications.* IEEE Press, 1994.

[62] M. Shaw and D. Garlan. *Software Architecture.* Prentice Hall, 1996.

[63] E. Simon. *Distributed Information Systems.* McGraw-Hill, 1996.

[64] M. Sloman, editor. *Network and Distributed Systems Management.* Addison-Wesley, 1994.

[65] M. Sloman and K. Twidle. Domains: A framework for structuring management policy. In Sloman [64], pages 433–453.

[66] M. Spreitzer and M. Theimer. Providing location information in a ubiquitous computing environment. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, volume 27 of *ACM SIGOPS*, pages 270–283, 1993.

[67] M. Spreitzer and M. Theimer. Scalable, secure, mobile computing with location information. *Communications of the ACM*, 36(7):27, 1993.

[68] M. Spreitzer and M. Theimer. Architectural considerations for scalable, secure, mobile computing with location information. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 29–38, Poznan, Poland, June 1994. IEEE CS Press.

[69] B. Sterzbach and W.A. Halang. A mobile vehicle on-board computing and communication system. *Computers and Graphics, Special Issue on Mobile Computing and Graphics*, 20(5), 1996.

[70] Sun Microsystems, Inc. *SunOS Reference Manual*, March 1990.

[71] C. D. Tomlin. *Geographic Information Systems and Cartographic Modeling.* Prentice Hall, 1990.

[72] K. Twidle. *Domain Services for Distributed Systems Management.* PhD thesis, Department of Computing, Imperial College, London, May 1993.

[73] J. Z. Wang. A fully distributed location registration strategy for universal personal communication systems. *IEEE Journal of Selected Areas in Communications*, 11(6), August 1995.

[74] R. Want, V. Falcao, and J. Gibbons. The Active Badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[75] R. Want and A. Hopper. Active Badges and personal interactive computing objects. *IEEE Transactions on Consumer Electronics*, 38(1), February 1992.

[76] M. Weiser. The computer for the 21st century. *Scientific American*, pages 66–75, September 1991.

[77] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7), 1993.

[78] D. Wells et al. *Guide to GPS Positioning.* Canadian GPS Associates, 1986.

[79] K. Wood, T. Richardson, F. Bennett, A. Harter, and A. Hopper. Global teleporting with Java: Towards ubiquitous personalized computing. *IEEE Computer*, 30(2):53–59, February 1997.

[80] M. Worboys. *GIS: A Computing Perpective.* Taylor & Francis, 1995.

[81] A. Yeo, A. Ananda, and E. Koh. A taxonomy of issues in name systems design and implementation. *Operating Systems Review*, 27(3):4–18, July 1993.

[82] W. Yeong, T. Howes, and S. Kille. RFC 1777. Lightweight Directory Access Protocol, March 1995.

[83] N. Yialelis. *Domain-Based Security for Distributed Object Systems.* PhD thesis, Department of Computing, Imperial College, London, August 1996.

[84] S. Zatti. Name management and directory services. In Sloman [64], pages 247–272.

[85] Y. Zhao. *Vehicle Location and Navigation Systems.* Intelligent Tranportation Systems. Artech House, 1997.